Volume 15 Number 1

ACTA CYBERNETICA

Szeged, 2001

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the University of Szeged, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 2001 Numbers 1–2 of Volume 15 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, University of Szeged, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-420-184, Fax:(36)-(62)-420-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

H. Bunke
Universität Bern
Institut für Informatik und
angewandte Mathematik
Längass strasse 51., CH-3012 Bern
Switzerland

B. Courcelle
Universitè Bordeaux-1
LaBRI, 351 Cours de la Libèration
33405 TALENCE Cedex
France

J. Demetrovics
MTA SZTAKI
Budapest, P.O.Box 63
H-1502 Hungary

B. Dömölki
IQSOFT
Budapest, Teleki Blanka u. 15-17.
H-1142 Hungary

J. Engelfriet
Leiden University
Computer Science Department
P.O. Box 9512, 2300 RA Leiden
The Netherlands

Z. Ésik
University of Szeged
Department of Foundations of
Computer Science
Szeged, Aradi vértanúk tere 1.
H-6720 Hungary

L. Lovász
Eötvös Loránd University
Budapest Múzeum krt. 6-8.
H-1088 Hungary

G. Păun
Institute of Mathematics
Romanian Academy
P.O.Box 1-764, RO-70700
Bucuresti, Romania

A. Prékopa
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

A. Salomaa
University of Turku
Department of Mathematics
SF-20500 Turku 50, Finland

L. Varga
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

H. Vogler
Dresden University of Technology
Department of Computer Science
Foundations of Programming
D-01062 Dresden, Germany

G. Wöginger
Technische Universität Graz
Institut für Mathematik (501B)
Steyrergasse 30
A-8010 Graz, Österreich

Acta Cybernetica 15 (2001) 1-8.

On some algebraic properties of automata*

András Ádám †

Abstract

Let $\mathcal{A}$ be a class of Moore automata. It is shown that $R(H(S(\mathcal{A})))$ is closed for the three operators S, H, R where S, H, R denote that the set of subautomata, of factor automata, of the automata obtained by input reduction (respectively) are formed.

Introduction

In the general theory of algebraic structures, the theorem of Tarski is one of the well-known results.[1] It concerns to how the narrowest class $V(\mathcal{A})$ can be produced from a class $\mathcal{A}$ of structures (all being of the same type) such that $V(\mathcal{A})$ is closed for the operators of forming direct products, subalgebras and factor algebras.

The present paper contains a variation on the theme of Tarski. We deal with automata (having output function) in the sense of Moore.[2] Our considerations concern to the operators of forming subautomata and factor automata, and to the operator of input reduction. (We study a weaker and a total version of the second and third of these operators.)

Let an arbitrary class $\mathcal{A}$ of finite Moore automata be considered. Let us denote by $K(\mathcal{A})$ the narrowest class which includes $\mathcal{A}$ and is closed for the three operators S, H, R mentioned above. Our main result expresses that $R(S(H(\mathcal{A})))$ exhausts the class $K(\mathcal{A})$. An auxiliary statement (Lemma 2) is now valid in a stronger form, than in the field of universal algebra (namely, equality can be asserted instead of set inclusion).

*Research partially supported by the Hungarian National Foundation for Scientific Research (OTKA) grant, no. T 026069.

†A. Rényi Mathematical Institute of the Hungarian Academy of Sciences, 1364 Budapest, P.O. Box 127, Hungary, email: tappancs@renyi.hu

[1] See [4] and Section 9 of Chapter 2 in [3].

[2] These automata cannot be regarded, in strict sense, to be algebraic structures. Although the transition function of an automaton may be viewed as a family consisting of $|X|$ unary operations, the output function is not an algebraic operation. In the field of automaton theory, direct products and substructures (we deal with the second of them only) have the same properties as familiar in algebra; the congruences and factor automata behave, however, somewhat curiously for an algebraist (cf. Sections 6–7 and Appendix 3 in [1]). The dissimilarity is continued when input reduction is considered; this operator does not preserve the type, in contrast to the usual algebraic operators which are type-preserving.

1 Basic terminology

By an *automaton* we mean a Moore-type automaton, written in form $\mathbf{A} = (A, X, Y, \delta, \lambda)$. (Here A, X, Y are nonempty finite sets.) The letter $\mathcal{A}$ is used for denoting a nonempty set consisting of automata. Isomorphic automata are regarded to be equal.

Some basic notions and facts of automaton theory are supposed to be known (see also Chapter 1 in [1]); including that there is a maximal congruence among the congruences of an automaton $\mathbf{A}$, and $a \equiv b \pmod{\pi_{\max}}$ precisely when the states a and b are indistinguishable (formally: when $\lambda(\delta(a,p)) = \lambda(\delta(b,p))$ for each input word p). We say that $\mathbf{A}$ is *simple* if $\pi_{\max}$ equals the trivial congruence of $\mathbf{A}$. Let π run through the congruences of $\mathbf{A}$, among the factor automata $\mathbf{A}/\pi$ solely $\mathbf{A}/\pi_{\max}$ is simple.

Let x_1, x_2 be input symbols, we say that x_1 and x_2 *act equally* (in $\mathbf{A}$) if $\delta(a, x_1) = \delta(a, x_2)$ for each $a(\in A)$. There is obviously a partition $\sigma_{\max}^{(A)}$ of the set X of input symbols such that the symbols being in a common partition class and only these act equally. $\mathbf{A}$ is called an *input-reduced* automaton if $\sigma_{\max}^{(A)}$ is the most refined partition of X (i.e., the partition whose index equals $|X|$). We can omit the superscript and write $\sigma_{\max}$ if there is no danger of confusion.

Let a partition $\sigma(\leq \sigma_{\max}^{(A)})$ of X be chosen. We can form the automaton $\mathbf{A}\backslash\sigma$ in a natural manner, namely, by identifying the input symbols which are in a common class mod σ.

2 The five operators

Consider an arbitrary class $\mathcal{A}$ of automata. Five operators will be introduced; by applying any of them, we obtain another automaton class from $\mathcal{A}$.

Let $\mathbf{D} \in S(\mathcal{A})$ hold when there is an $\mathbf{A}(\in \mathcal{A})$ such that $\mathbf{D}$ is a subautomaton of $\mathbf{A}$.

Let $\mathbf{C} \in H(\mathcal{A})$ hold when there are an $\mathcal{A}(\in \mathcal{A})$ and a congruence π of $\mathbf{A}$ such that $\mathbf{C} = \mathbf{A}/\pi$.

Let $\mathbf{C}_1 \in H^{\Delta}(\mathcal{A})$ hold when $\mathbf{C}_1$ is simple and $\mathbf{C}_1 \in H(\mathcal{A})$.

Let $\mathbf{B} \in R(\mathcal{A})$ hold when there are an $\mathbf{A}(\in \mathcal{A})$ and a partition σ of the set X of input symbols or $\mathbf{A}$ such that ($\sigma \leq \sigma_{\max}^{(A)}$ and) $\mathbf{B} = \mathbf{A}\backslash\sigma$.

Let $\mathbf{B}_1 \in R^{\Delta}(\mathcal{A})$ hold when $\mathbf{B}_1$ is input-reduced and $\mathbf{B}_1 \in R(\mathcal{A})$.

In the final part of this section some evident consequences of the definitions above are listed.

Denote by Q any of $S, H, H^{\Delta}, R, R^{\Delta}$. The equality $Q(Q(\mathcal{A})) = Q(\mathcal{A})$ holds (i.e., the operators are idempotent), and

$$Q(\mathcal{A}) = \cup_{\mathbf{A} \in \mathcal{A}} Q(\mathbf{A}).$$

In case $|\mathcal{A}| = 1$ we write $Q(\mathbf{A})$ instead of $Q(\{\mathbf{A}\})$.

It is clear that the equalities

$$H^{\Delta}(H(\mathcal{A})) = H(H^{\Delta}(\mathcal{A})) = H^{\Delta}(\mathcal{A}) \tag{2.1}$$

and

$$R^{\Delta}(R(\mathcal{A})) = R(R^{\Delta}(\mathcal{A})) = R^{\Delta}(\mathcal{A}), \tag{2.2}$$

furthermore, the inclusions

$$S(\mathcal{A}) \supseteq \mathcal{A}, \;\; H(\mathcal{A}) \supseteq \mathcal{A}, \;\; R(\mathcal{A}) \supseteq \mathcal{A} \tag{2.3}$$

are valid. The membership relations $\mathbf{D} \in S(\mathbf{A})$ and $\mathbf{C} \in H(\mathbf{A})$ imply

$$\sigma^{(A)}_{\max} \leq \sigma^{(D)}_{\max}, \;\; \sigma^{(A)}_{\max} \leq \sigma^{(C)}_{\max}, \tag{2.4}$$

respectively.

3 The main result

Now we can expose the Tarski-type statements for automata.

Theorem 1 *Let a class $\mathcal{A}$ of automata be considered. Denote by $\mathcal{K}$ the narrowest class such that $\mathcal{K} \supseteq \mathcal{A}$ and $\mathcal{K}$ is closed for the operators S, H, R.*

(I) *$\mathcal{K}$ equals $R(H(S(\mathcal{A})))$.*

(II) *The class of the simple automata belonging to $\mathcal{K}$ equals $R(H^{\Delta}(S(\mathcal{A})))$.*

(III) *The class of the input-reduced automata belonging to $\mathcal{K}$ equals $R^{\Delta}(H(S(\mathcal{A})))$.*

(IV) *The class of the input-reduced simple automata belonging to $\mathcal{K}$ equals $R^{\Delta}(H^{\Delta}(S(\mathcal{A})))$.* □

4 Proof of the main result

The following facts can be seen easily:

Lemma 1 *The operator R does not alter the distinguishability of states of an automaton. Hence the subsequent three conditions are equivalent for an automaton $\mathbf{A}$:*

(i) *$\mathbf{A}$ is simple.*

(ii) *$R(\mathbf{A})$ contains at least one simple automaton.*

(iii) *All the automata belonging to $R(\mathbf{A})$ are simple.*

□

Lemma 2 $S(H(\mathcal{A})) = H(S(\mathcal{A}))$.

Proof. Assume $\mathbf{A} \in \mathcal{A}$ and $\mathbf{D} \in S(H(\mathbf{A}))$. Then there exist an automaton $\mathbf{C}(\supseteq \mathbf{D})$ and a homomorphism χ such that χ maps $\mathbf{A}$ onto $\mathbf{C}$. The states a of $\mathbf{A}$ for which $\chi(a)$ belongs to the state set of $\mathbf{D}$ constitute a subautomaton $\mathbf{D}'$ of $\mathbf{A}$. It is obvious that $\mathbf{D}$ is the image of $\mathbf{D}'$ under the appropriate restriction of χ.

Conversely, suppose $\mathbf{A} \in \mathcal{A}$ and $\mathbf{C} \in H(S(\mathbf{A}))$. There exist a subautomaton $\mathbf{D}$ of $\mathbf{A}$ and a congruence π_1 of $\mathbf{D}$ such that $\mathbf{D}/\pi_1$ and $\mathbf{C}$ are isomorphic. Introduce a partition π_2 of the state set of $\mathbf{A}$ such that

(α) the restriction of π_2 to the state set of $\mathbf{D}$ coincides with π_1, and

(β) any state of $\mathbf{A}$ which is not contained in $\mathbf{D}$ forms a one-element class mod π_2.

It is evident that π_2 is a congruence of $\mathbf{A}$ and $\mathbf{D}/\pi_1$ is a subautomaton of $\mathbf{A}/\pi_2$. □

Lemma 3 $S(R(\mathcal{A})) \subseteq R(S(\mathcal{A}))$.

Proof. Suppose $\mathbf{A} \in \mathcal{A}$ and $\mathbf{D} \in S(R(\mathbf{A}))$. Then there exist a $\mathbf{B}(\supseteq \mathbf{D})$ and a partition σ of X such that $\mathbf{B} = \mathbf{A}\backslash\sigma$. We have clearly $\mathbf{D} = \mathbf{D}_1\backslash\sigma$ where $\mathbf{D}_1$ is a subautomaton of $\mathbf{A}$ such that the state sets of $\mathbf{D}_1$ and $\mathbf{D}$ coincide.

Lemma 4 $H(R(H(\mathcal{A}))) = R(H(\mathcal{A}))$.

Proof. The inclusion $\supseteq$ holds by (2.3). It suffices to show the relation $\subseteq$ when $\mathcal{A} = \{\mathbf{A}\}$.

Assume $\mathbf{C}_1 \in H(R(H(\mathbf{A})))$. This supposition means the existence of two automata $\mathbf{C}_2, \mathbf{B}$, a partition σ of X, two homomorphisms χ_1, χ_2 such that

$$\mathbf{B} \in R(H(\mathbf{A})), \quad \mathbf{C}_2 \in H(\mathbf{A}), \quad \mathbf{B} = \mathbf{C}_2\backslash\sigma,$$

moreover, χ_1 maps $\mathbf{B}$ onto $\mathbf{C}_1$ and χ_2 maps $\mathbf{A}$ onto $\mathbf{C}_2$. The state sets of $\mathbf{C}_2$ and $\mathbf{B}$ are equal.

Denote the kernels of χ_2 and χ_1 by π_2 and π_1, respectively. (π_2 is a congruence of $\mathbf{A}$, π_1 is a congruence of $\mathbf{B}$ as well as of $\mathbf{C}_2(= \mathbf{A}/\pi_2)$.) Introduce a partition π_1' of the state set A of $\mathbf{A}$ by

$$a \equiv b \pmod{\pi_1'} \Leftrightarrow \chi_2(a) \equiv \chi_2(b) \pmod{\pi_1}.$$

π_1' is a congruence of $\mathbf{A}$ (since π_1, π_2 are congruences), and

$$\mathbf{C}_1 = (\mathbf{A}/\pi_1')\backslash\sigma.$$

This representation of $\mathbf{C}_1$ assures $\mathbf{C}_1 \in R(H(\mathbf{A}))$. □

Lemma 5 $H^{\Delta}(R(\mathcal{A})) \subseteq R(H^{\Delta}(\mathcal{A}))$.

Proof. As in the preceding proof, we deal with the case $\mathcal{A} = \{\mathbf{A}\}$. Let $\mathbf{C}$ belong to $H^{\Delta}(R(\mathbf{A}))$. There exists a $\sigma(\leq \sigma_{\max}^{(A)})$ such that, with the maximal congruence π of $\mathbf{B} = \mathbf{A}\backslash\sigma$, we have $\mathbf{C} = \mathbf{B}/\pi$. The first sentence of Lemma 1 guarantees that the state partition π is the maximal congruence of A also, thus $\mathbf{C}_1 = \mathbf{A}/\pi$ is a simple automaton. There exists the automaton $\mathbf{C}_1\backslash\sigma$ (since $\sigma \leq \sigma_{\max}^{(C_1)}$ by (2.4)), and

$$\mathbf{C} = \mathbf{C}_1\backslash\sigma \in R(H^{\Delta}(\mathbf{A}))$$

is evident. □

Lemma 6 $H^{\Delta}(R(H(\mathcal{A}))) = R(H^{\Delta}(\mathcal{A}))$.

Proof. Assume $\mathbf{C}_2 \in R(H^{\Delta}(\mathcal{A}))$. The automaton $\mathbf{C}_2$ is simple (by the second sentence of Lemma 1), consequently

$$\{\mathbf{C}_2\} = H^{\Delta}(\mathbf{C}_2) \subseteq$$

$$\subseteq H^{\Delta}(R(H^{\Delta}(\mathcal{A}))) \subseteq H^{\Delta}(R(H(\mathcal{A}))).$$

Thus $\supseteq$ has been verified. The inclusion $\subseteq$ follows from Lemma 5 and (2.1):

$$H^{\Delta}(R(H(\mathcal{A}))) \subseteq R(H^{\Delta}(H(\mathcal{A}))) = R(H^{\Delta}(\mathcal{A})).$$

□

Proof of Theorem 1. For verifying (I), first we observe

$$(\mathcal{A} \subseteq)R(H(S(\mathcal{A}))) \subseteq \mathcal{K}.$$

Conversely, suppose $\mathbf{B} \in \mathcal{K}$. There exist a positive integer t and t automata $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t$ such that $\mathbf{A}_1 \in \mathcal{A}, \mathbf{A}_t = \mathbf{B}$, and, for any i (where $2 \leq i \leq t$), $\mathbf{B}_i$ can be obtained from $\mathbf{B}_{i-1}$ either by R or by H or by S.

Our next aim is to show the implication

$$\mathbf{A}_{i-1} \in R(H(S(\mathbf{A}_1))) \Rightarrow \mathbf{A}_i \in R(H(S(\mathbf{A}_1))). \tag{4.1}$$

If $\mathbf{A}_i \in R(\mathbf{A}_{i-1})$ or $\mathbf{A}_i \in H(\mathbf{A}_{i-1})$, then (4.1) holds by the idempotency of R or by Lemma 4, respectively. When $\mathbf{A}_i \in S(\mathbf{A}_{i-1})$, then (4.1) follows from Lemmas 2, 3 and the idempotency of S.

Our inference can be summarized as follows:

$$\mathbf{B} \in \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t\} \subseteq R(H(S(\mathbf{A}_1))) \subseteq R(H(S(\mathcal{A}))).$$

Now we turn to showing (II) and (III). Lemma 6 and (2.2) imply the equalities

$$H^{\Delta}(R(H(S(\mathcal{A})))) = R(H^{\Delta}(S(\mathcal{A}))) \tag{4.2}$$

and

$$R^{\Delta}(R(H(S(\mathcal{A})))) = R^{\Delta}(H(S(\mathcal{A}))), \tag{4.3}$$

respectively. Since the statement (I) is true, (4.2) expresses (II) and (4.3) expresses (III).

Finally, we prove (IV). Consider an arbitrary input-reduced simple automaton $\mathbf{B}'$which is contained in $\mathcal{K}$. We have

$$\{\mathbf{B}'\} = R^{\Delta}(H^{\Delta}(\mathbf{B}')) \subseteq R^{\Delta}(H^{\Delta}(\mathcal{K})).$$

This means that $R^{\Delta}(H^{\Delta}(\mathcal{K}))$ exhausts the class of input-reduced simple automata which belong to $\mathcal{K}$. The deduction

$$R^{\Delta}(H^{\Delta}(\mathcal{K})) = R^{\Delta}(H^{\Delta}(R(H(S(\mathcal{A}))))) =$$

$$= R^{\Delta}(R(H^{\Delta}(S(\mathcal{A})))) = R^{\Delta}(H^{\Delta}(S(\mathcal{A})))$$

is valid by (I), Lemma 6 and (2.2). □

5 Final remarks

Some statements, related to lemmas in the preceding section, can be proved by similar ideas; for example, the equalities

$$H^{\Delta}(S(H(\mathcal{A}))) = S(H^{\Delta}(\mathcal{A}))$$

and

$$H^{\Delta}(R^{\Delta}(H^{\Delta}(\mathcal{A}))) = R^{\Delta}(H^{\Delta}(\mathcal{A})). \tag{5.1}$$

I have become acquainted with facts belonging to the present topics when H. Andréka and Zs. Baranyai showed that (5.1) holds (in case $|\mathcal{A}| = 1$) but

$$R^{\Delta}(H^{\Delta}(R^{\Delta}(\mathcal{A}))) = H^{\Delta}(R^{\Delta}(\mathcal{A}))$$

is not valid in general [2].

We have stated *equality* in Lemma 2 for automata, in the general theory of algebraic structures only the *inclusion* $S(H(\mathcal{A})) \subseteq H(S(\mathcal{A}))$ is valid. In addition, it follows from Lemma 2 that

$$S(H^{\Delta}(\mathcal{A})) = H^{\Delta}(S(H(\mathcal{A}))) = H^{\Delta}(H(S(\mathcal{A}))) = H^{\Delta}(S(\mathcal{A})) \tag{5.2}$$

in the field studied here. Consequently, the formulae in the statements (I)–(IV) of Theorem 1 can equivalently be replaced by

$$R(S(H(\mathcal{A}))), \quad R(S(H^{\Delta}(\mathcal{A}))),$$

$$R^{\Delta}(S(H(\mathcal{A}))), \quad R^{\Delta}(S(H^{\Delta}(\mathcal{A}))),$$

respectively.

a	$\delta(a, x_1)$	$\delta(a, x_2)$
1	2	3
2	3	3
3	2	2

Table 1

e	$\delta(e, x_1)$	$\delta(e, x_2)$	$\lambda(e)$
1	2	4	y_1
2	3	3	y_2
3	1	1	y_3
4	3	3	y_2

(a)

c	$\delta(c, x_1)$	$\delta(c, x_2)$	$\lambda(c)$
1	2	2	y_1
2	3	3	y_2
3	1	1	y_3

(b)

d	$\delta(d, x)$	$\lambda(d)$
1	2	y_1
2	3	y_2
3	1	y_3

(c)

Table 2

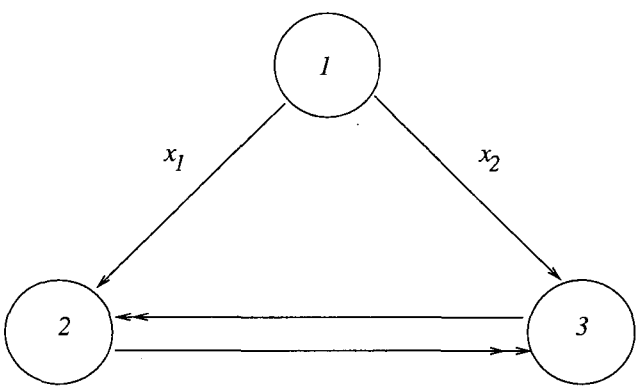


Figure 1

Is Lemma 3 true with equality (instead of $\subseteq$)? The next example shows that the answer is negative (in general). Consider the automaton $\mathbf{A}$ determined by Table 1 (the output function is indifferent), see also Figure 1. Let $\mathbf{B}$ be the autonomous automaton having two states in which $\delta(b_1, x) = b_2$ and $\delta(b_2, x) = b_1$. This $\mathbf{B}$ is contained in $R(S(\mathbf{A}))$, it does not belong to $S(R(\mathbf{A}))$.

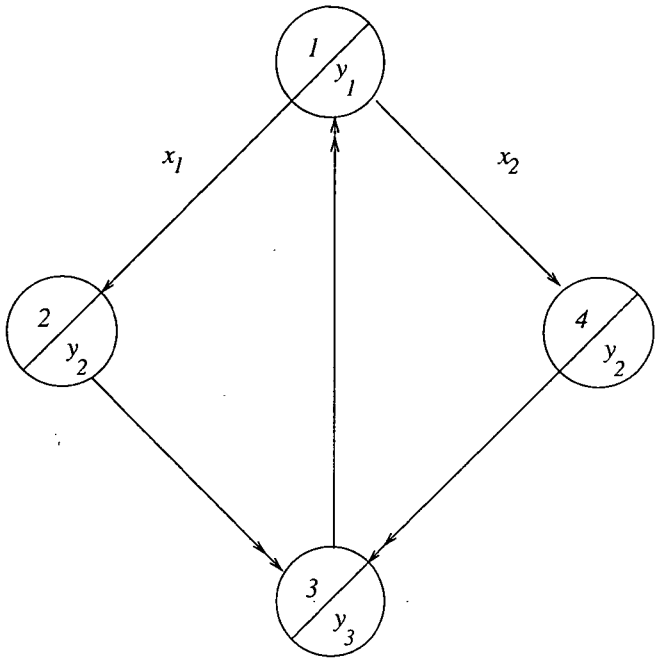


Figure 2

Analogously, Lemma 5 loses its validity if inclusion is replaced by equality. Indeed, let $\mathbf{E}, \mathbf{C}, \mathbf{D}$ be the Moore automata determined by Tables 2/a, 2/b, 2/c, respectively; see also Figure 2 for $\mathbf{E}$. Then $R(H^{\Delta}(\mathbf{E})) = \{\mathbf{C}, \mathbf{D}\}$ and $H^{\Delta}(R(\mathbf{E})) = \{\mathbf{D}\}$.

References

[1] Ádám, A., *The behaviour and simplicity of finite Moore automata*, Akadémiai Kiadó, Budapest, 1996.

[2] Andréka, H. and Baranyai, Zs., Personal communication (in the late 1960's).

[3] Burris, S. and Sankappanavar, H. P., *A course in Universal Algebra*, Springer, New York, 1981. (Hungarian translation: Tankönyvkiadó, Budapest, 1988.)

[4] Tarski, A., A remark on functionally free algebras, *Annals of Math.* 47 (1946), 163-165.

Received September, 2000

Acta Cybernetica **15** (2000) 9-23.

P Systems with Communication Based on Concentration

Jürgen Dassow * Gheorghe Păun †

Abstract

We consider a variant of P systems where the communication of objects is controlled by the "concentration" of these objects: after each evolution step, the objects are redistributed among the regions of the system in such a way that each region contains the same number of copies of each object (plus/minus one, when the number of objects is not divisible by the number of regions). We show that P systems of this form, with only one flip-flop catalyst but without using other control ingredients, can generate the Parikh images of all matrix languages. When an unbounded number of catalysts is available, a characterization of recursively enumerable sets of vectors of natural numbers is obtained (by systems with only one membrane).

1 Introduction

The P systems are a class of distributed parallel computing devices of a biochemical type (so, they belong to the rather active area of Molecular Computing) which were recently introduced in [11]; an early survey can be found in [12].

In short, in the basic model one considers a *membrane structure* consisting of several cell-membranes which are hierarchically embedded in a main membrane, called the *skin* membrane. The membranes delimit *regions*, where we place *objects*, elements of a finite set (an alphabet). The objects evolve according to given *evolution rules*, which are associated with the regions. An object can evolve independently of the other objects in the same region of the membrane structure, or in cooperation with other objects. In particular, we can consider *catalysts*, objects which do not evolve alone, but only assist other objects to evolve. The evolution rules are given in the form of transition rules for multisets, can be the subject of a given priority relation, and in their right hand members contain symbols $(a, here), (a, out), (a, in_j)$, where a is an object. The meaning is that one

*Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, PSF 4120, D-39016 Magdeburg, Germany, E-mail: dassow@iws.cs.uni-magdeburg.de

†Corresponding author. Research supported by Alexander von Humboldt Foundation. Institute of Mathematics of the Romanian Academy, PO Box 1 - 764, 70700 Bucureşti, Romania, E-mail: gpaun@imar.ro

occurrence of the symbol a is produced and remains in the same region, is sent out of the current region, or is sent to the region associated with membrane j (which should be reachable from the region where the rule is applied), respectively. The membranes can be *dissolved*. When such an action takes place, all the objects of the dissolved membrane remain free in the membrane placed immediately outside, but the evolution rules of the dissolved membrane are lost. The skin membrane is never dissolved.

The application of evolution rules is done in a maximally parallel manner: at each step, all objects which can evolve should evolve.

Starting from an initial configuration and using the evolution rules, we get a *computation*. We consider a computation completed when it halts, no further rule can be applied. The multiplicity of objects present in a designated membrane in a halting configuration is the result of the computation. Thus, in this way we compute vectors of natural numbers.

Many variants are considered in [3], [4], [5], [10], [11], [13], [14], [16]. Most of them are computationally complete, equal in power to Turing machines. When an enhanced parallelism is provided, for instance, by allowing membrane division, then linear time solutions to NP-complete problems can be obtained, [7], [10], [14], [19].

One of the most non-realistic features of many of these variants is the use of the target indications *out* and in_j; expecially the last one is rather far from biochemistry. Attempts to get rid of indications of the form in_j were already done in [13] (where electrical charges are used instead of labels: each object is marked with $+, -$, or 0 and the same with the membranes; when an object is introduced with a mark $+$ or $-$, it will be passed to a membrane of the opposite sign, nondeterministically chosen among the neighboring membranes) and in [15] (indications of the form *in*, without any membrane specification, are used, associated with other ingredients which are used to control the communication).

Here we make one more step "towards reality": the main way of communicating chemical objects in biochemistry is based on differences of concentration (gradient) among regions of a cell, see [8], [9]. We consider here a variant of P systems where this idea alone governs all communications (that is, we remove all commands *out* and in_j, and we use only communication driven by the concentration difference). We use no other control ingredients (priority among evolution rules, actions controlling the thickness of membranes, such as the dissolving action, etc), but only catalysts (in the powerful form of bistable catalysts, able to change their state among two states, in a flip-flop manner). Using only one catalyst, we cover in this way at least the Parikh images of matrix languages. When an arbitrary number of bistable catalysts is used, we can characterize the recursively enumerable sets of vectors of natural numbers. Systems consisting of one membrane only are enough (hence in such a case only the communication to the outer region of the system is possible).

2 A Remark on Matrix Grammars

Before we consider P systems we briefly recall the definition of matrix grammars and their associated languages because we shall use those grammars in the study of the generative power of P systems. Moreover, we add a new normal form for matrix grammars which is useful in this paper and is of some interest in the theory of matrix grammars. For further elements of formal language theory we refer to [17]. (We only mention that V^* is the free monoid generated by V under the operation of concatenation; λ is the empty string, $|x|$ is the length of $x \in V^*$ and $\Psi_V(x)$ is the Parikh vector associated with $x \in V^*$.)

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (we say that N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT. By $PsMAT$ we denote the Parikh sets associated with languages in MAT.

It is known that $MAT \subset MAT_{ac} = RE$ and that each one-letter language in the family MAT is regular, [6]. Further details about matrix grammars can be found in [2] and in [17].

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets being mutually disjoint, and the matrices in M are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in N_2^*, |x| \leq 2$, or $x \in T \cup \{\lambda\}$,
3. $(X \to Y, A \to \dagger)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T \cup \{\lambda\}$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3; $\dagger$ is a trap symbol, once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

Furthermore, for any symbol $X \in N_1$, there is a matrix whose left-hand side of the first rule is X.

According to Lemma 1.3.7 in [2], for each matrix grammar there is an equivalent matrix grammar in binary normal form.

We now introduce a stronger normal form.

A matrix grammar $G = (N, T, S, M, F)$ is said to be in *strong binary normal form*, if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets being mutually disjoint, and the matrices in M are of one of the following forms:

1. $(S \to XA)$ with $X \in N_1$, $A \in N_2$,
2. $(X \to Y,\ A \to \alpha)$ with $X, Y \in N_1$, $A \in N_2$, $\alpha \in N_2^2 \cup N_2 \cup T \cup \{\lambda\}$,
3. $(X \to Y,\ A \to \dagger)$ with $X, Y \in N_1$, $A \in N_2$,
4. $(X \to \lambda)$ with $X \in N_1$.

Moreover, there is only one matrix of type 1 (the start matrix) and only one of type 4 (the final matrix) and F consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3, where $\dagger$ is the trap symbol, again. The matrix of type 4 is used only once, in the last step of a derivation such that after the application of this final matrix only terminal symbols and possibly some trap symbols are left. Finally, we may assume that each control symbol from N_1 appears at least once on the left-hand side of the first production of a matrix in M and that the final control symbol only appears on the left-hand side of the final production.

Lemma 1. *For any matrix grammar G with appearance checking there is a matrix grammar G' in strong binary normal form such that $L(G') = L(G)$.*

Proof. Let $G = (N, T, S, M, F)$ be a matrix grammar. Without loss of generality, we can assume that G is in binary normal form as given above. Thus $N = N_1 \cup N_2 \cup \{S, \dagger\}$. Let n be the cardinality of N_2 and $N_2 = \{A_k \mid 1 \leq k \leq n\}$. We immediately obtain the corresponding matrix grammar in strong binary normal form $G' = (N', T, S, M', F')$ in the following way:

- $N' = N \cup \{Z_k \mid 0 \leq k \leq n\}$, where the Z_k, $0 \leq k \leq n$, are new symbols not contained in N.
- $M' = (M \setminus \{(X \to \lambda,\ A \to \alpha) \mid (X \to \lambda,\ A \to \alpha) \in M\}) \cup M''$ with
  $M'' = \{(X \to Z_0, A \to \alpha) \mid (X \to \lambda, A \to \alpha) \in M\} \cup$
  $\{(Z_{k-1} \to Z_k,\ A_k \to \dagger) \mid 1 \leq k \leq n\} \cup \{(Z_n \to \lambda)\}$,
- $(Z_n \to \lambda)$ is the final matrix.

Obviously, instead of one of the final matrices $(X \to \lambda,\ A \to \alpha)$ of G we use $(X \to Z_0,\ A \to \alpha)$, check the presence of an element of N_2 by the rules $(Z_{k-1} \to Z_k,\ A_k \to \dagger)$, $1 \leq k \leq n$, and finally terminate by $(Z_n \to \lambda)$. □

3 P Systems: Some Variants

We first introduce the basic class of P systems, then we briefly define several variants. The definitions are not completely formal; for details, the reader is referred to the papers listed in the bibliography.

A *membrane structure* is a construct consisting of several *membranes* placed in a unique "skin" membrane; we identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. Graphically, a membrane structure is represented by a Venn diagram.

Each membrane identifies a *region*, delimited by it and the membranes immediately inside it (if any). A membrane without any other membrane inside it is said to be *elementary*.

Figure 1 represents a membrane structure, described by the string

$$\mu = [_1[_2[_3[_4\,]_4]_3[_5\,]_5]_2[_6[_7\,]_7[_8\,]_8]_6[_9\,]_9]_1.$$

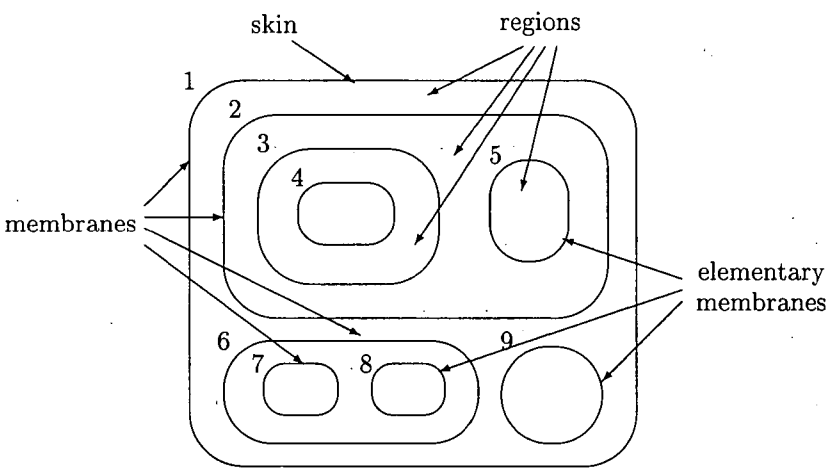


Figure 1. A membrane structure.

If in the regions delimited by the membranes we place multisets of objects from a specified finite set V, as well as evolution rules for these objects, then we obtain a *P system.*

More precisely, a P system (of degree $m, m \geq 1$) is a construct

$$\Pi = (V, T, C, \mu, w_1, \ldots, w_m, R_1, , \ldots, R_m),$$

where:

(i) V is an alphabet; its elements are called *objects*;

(ii) $T \subseteq V$ (the *output* alphabet);

(iii) $C \subseteq V, C \cap T = \emptyset$ (*catalysts*);

(iv) μ is a membrane structure consisting of m membranes (labeled with $1, 2, \ldots, m$);

(v) $w_i, 1 \leq i \leq m$, are strings over V associated with the regions $1, 2, \ldots, m$ of μ; they represent multisets of objects present in the regions of μ (the multiplicity of a symbol in a region is given by the number of occurrences of this symbol in the string corresponding to that region);

(vi) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over V associated with the regions $1, 2, \ldots, m$ of μ.
The evolution rules are of the forms $a \to v$ or $ca \to cv$, where $a \in V - C$, $c \in C$, and v is a string over

$$(V \times \{here, out\}) \cup (V \times \{in_j \mid 1 \leq j \leq m\}).$$

When presenting the evolution rules, the indication "here" is often omitted.

The membrane structure and the multisets w_i, $1 \leq i \leq m$, in Π constitute the *initial configuration* of the system. We can pass from a configuration to another one by using the evolution rules. This is done in parallel: all objects, from all membranes, that can be the subject of local evolution rules, should evolve simultaneously.

The use of a rule $ca \to cv$ (similarly for rules $a \to v$) in a region with a multiset w means to subtract the multiset a from w, then to follow the prescriptions of v: if an object appears in v in the form $(b, here)$, then it remains in the same region; if we have (b, out), then a copy of the object b will be introduced in the membrane immediately outside the region of the rule $ca \to cv$ (if the rule is applied in the skin membrane, then the object leaves the system); if we have (b, in_i), then a copy of b is introduced in the membrane with the label i, providing that this membrane is adjacent to the region of the rule $ca \to cv$, otherwise the rule cannot be applied

A sequence of transitions between configurations of a given P system Π is called a *computation* with respect to Π. A computation is *successful* if and only if it halts, that is, there is no rule applicable to the objects present in the last configuration. The result of a successful computation consists of the vector of natural numbers which specify the number of copies of terminal objects which leave the skin membrane (e.g., if $T = \{a, b, c\}$ and no copy of a, four copies of b, and two copies of c leave the system, then the computed vector is $(0, 4, 2)$.)

Several further ingredients were considered in the literature: priority relations among evolution rules, the action of dissolving a membrane, an opposite action to dissolving, which makes thicker the membranes, inhibiting the communication, "electrical charges" associated with objects and membranes, used for controlling the communication [13], bistable catalysts [16]. We introduce here only the last of these ingredients, because it will be also used in our variant of P systems.

The bistable catalysts are catalysts able to change their state among two possibilities, c and $\bar{c}$. Specifically, we allow rules of the forms $ca \to \bar{c}v$ and $\bar{c}a \to cv$, but not also of the forms $ca \to cv$ and $\bar{c}a \to \bar{c}v$; that is why we also call these catalysts *flip-flop catalysts.*

For the power of P systems which use certain combinations of these ingredients we refer to the papers mentioned at the end of the present work. In particular, several characterizations of the recursively enumerable sets of vectors of natural numbers can be found in [11], [13], [15].

4 Definition of P Systems with Communication Based on Concentration

The indication in_i looks rather unrealistic from a biochemical point of view, so it is of interest to try to avoid its use. Actually, we will remove all indications *here, out,* in_i, by controlling the communication of objects only by means of their concentration in the regions of the system.

At the first sight, this can be done in a very simple way: in the set V of objects we consider a subset, $V_c \subseteq V$, and whenever such objects are introduced, they are immediately redistributed among all regions of the system in such a way that all regions will have the same number of occurrences of each symbol from V_c; a difference of one occurrence is allowed when the total number of occurrences is not divisible by the number of regions. Note that a system with m membranes defines $m + 1$ regions, m regions inside the system, plus the outer region.

Note also that always we take into consideration each object in V_c separately and we do not count these objects (and then redistribute them) as being indistinguishable.

A problem appears when the number of copies of one object to be redistributed is not a multiple of $m + 1$, where m is the number of membranes in the system. In such a case, the "remainder objects" are redistributed according to the following "efficiency principle": we move objects at the lowest distance (crossing the smallest number of membranes). For instance, if we have $m + 2$ objects in a region, then all regions (including the outer region) will get one object, while the region where the objects were produced will remain with two copies. If we have started with $m + 3$ objects, then each region will get one object, one further object will remain in the region where the objects were produced, and one further object will be sent to one of the neighboring regions, randomly choosen.

Because in the proofs in the following sections we will always introduce objects to be communicated in only one region at a time, the above mentioned principle will be enough to govern the communication, so we ignore the more complex situations which can appear.

In this framework, no indication *here, in, out* is necessary. The objects in V_c will go in or out, according to their concentration, the objects in $V - V_c$ remain in the region where they are introduced.

Of course, no catalyst can appear in V_c and all terminal symbols are in this set (we have to read the result of a computation outside the system, hence we have to send out of the system the terminal symbols).

How the redistribution of objects takes place in a practical circumstance is not considered here. We assume that the redistribution is done instantaneously, always correctly, and at the level of the whole system in a step, by a magical mechanism which is not explicit in our model.

We denote by $Ps(\Pi)$ the set of vectors of natural numbers computed by a P system Π and by $PsLP_m(con, 2Cat_k)$ the family of sets $Ps(\Pi)$ of this form generated by P systems which communicate by concentration, use at most k bistable catalysts, and have at most m membranes; when m or k are not specified, we replace them with $*$. (The notation reminds the notion of a *Parikh set*, $\Psi_V(L)$, associated with a language $L \subseteq V^*$.) We also denote by $PsRE$ the Parikh images of recursively enumerable languages (the family of such languages is denoted by RE), which is nothing else than the family of recursively enumerable sets of vectors of natural numbers.

5 The Power of P Systems with Concentration-Based Communication

First we show that systems with one pair of catalysts are sufficient to generate all matrix languages (without appearance checking).

Theorem 1. $PsMAT \subset PsLP_*(con, 2Cat_1)$.

Proof. We first prove the inclusion.

Let us consider a matrix grammar without appearance checking, $G = (N, T, S, M)$, in the binary normal form (that is, $N = N_1 \cup N_2 \cup \{S\}$). Assume that it contains n two-rule matrices, labeled in a one-to-one manner with $m_1, m_2, \ldots, m_n$.

We construct the P system (of degree $n+1$)

$$\Pi = (V, V_c, T, C, \mu, w_0, w_1, \ldots, w_n, R_0, R_1, \ldots, R_n),$$

with

$$\begin{aligned}
V &= N \cup T \cup V_c \cup C \cup \{D, E, E', E'', \bar{E}, \dagger, \#\} \cup \{X' \mid X \in N_1\},\\
V_c &= \{X_i, A_i \mid m_i = (X \to \alpha, A \to \beta) \in M, 1 \le i \le n\}\\
&\cup \{\bar{X} \mid X \in N_1\} \cup T \cup \{\bar{a} \mid a \in T\},\\
C &= \{c, \bar{c}\},\\
\mu &= [_0[_1\,]_1[_2\,]_2 \cdots [_n\,]_n]_0,\\
w_0 &= XADEc, \text{ for } (S \to XA) \text{ being the initial matrix of } G,\\
w_i &= Ec, \text{ for all } i = 1, 2, \ldots, n,
\end{aligned}$$

and with the following sets of evolution rules:

1. The set R_0 contains the following rules:
   (a) $X \to X_i^{n+2}$,
   $cA \to \bar{c}A_i^{n+2}$, for all $1 \leq i \leq n$ such that $m_i : (X \to \alpha, A \to \beta) \in M$,
   (b) $cD \to \bar{c}\dagger$,
   $\dagger \to \dagger$,
   (c) $\bar{c}\bar{X} \to cX$, for all $X \in N_1$,
   (d) $\bar{c}E'' \to c\bar{E}$,
   (e) $E \to E'$,
   $E' \to E''$,
   $E'' \to E$,
   (f) $\bar{a} \to a^{n+2}$,
   $a \to \#$, for all $a \in T$,
   (g) $X_i \to \#$, for all $X \in N_1, i = 1, 2, \ldots, n$,
   $A_i \to \#$, for all $A \in N_2, i = 1, 2, \ldots, n$.
2. For each $i = 1, 2, \ldots, n$ such that $m_i : (X \to \alpha, A \to \beta)$, the set R_i contains the following rules:
   (a) $cX_i \to \bar{c}\bar{\alpha}^2$
   (of course, if $\alpha = \lambda$, then the rule is $cX_i \to \bar{c}$),
   (b) $\bar{c}A_i \to c\bar{\beta}^2$,
   where $\bar{\beta}$ is $\bar{a}$ for $\beta = a, a \in T$ and $\bar{\beta} = \beta$ if $\beta \in N_2^*$,
   (c) $cA_i \to \bar{c}\dagger$,
   $\bar{c}E \to c\dagger$,
   $\dagger \to \dagger$,
   (d) $Y_j \to \#$, for all $Y \in N_1, 1 \leq j \leq n, j \neq i$,
   $B_j \to \#$, for all $B \in N_2, 1 \leq j \leq n, j \neq i$,
   (e) $\bar{Y} \to \#$, for all $Y \in N_1$,
   $B \to \#$, for all $B \in N_2$,
   $a \to \#$,
   $\bar{a} \to \#$, for all $a \in T$.

The system works as follows.

The object $\dagger$ is a trap object, once introduced it can evolve forever, thus the computation will never finish. The object $\#$ is a dummy one, non-active.

Each matrix from M is simulated in Π in three steps, as follows.

Assume that in the skin membrane we have one symbol from N_1, some symbols from N_2, the symbols D, E, and the catalyst c (plus occurrences of the dummy symbol, which we will ignore from now on); initially, we have here the multiset represented by $XADEc$, where $(S \to XA)$ is the initial matrix of M. Assume that at the same time in each membrane $1, 2, \ldots, n$ we have the objects E and c, as in the initial stage.

In the skin membrane, the symbol $X \in N_2$ will introduce $n+2$ copies of X_i, for some $1 \leq i \leq n$, and, at the same time, the catalyst plus a symbol $A \in N_2$ will introduce $n+2$ symbols A_j, for some $1 \leq j \leq n$, while the catalyst becomes $\bar{c}$; moreover, E will be replaced by E'. The symbols X_i, A_j should be redistributed. Because we have exactly $n+2$ copies of each of them, exactly one copy of each of them will be placed in each region of the system (one symbol leaves the system). Note that if the catalyst is not used by a rule $cA \to \bar{c}A_j^{n+2}$, then it must be used by the rule $cD \to \bar{c}\dagger$ and the computation never stops.

We distinguish eight cases, according to the symbols present in a membrane $i, 1 \leq i \leq n$, after one evolution step (we denote by z_i the string representing the multiset of these symbols):

1. $z_i = X_iA_iEc$. Using the rule $cX_i \to \bar{c}\bar{\alpha}^2$ we pass to (the multiset represented by) $z' = \bar{\alpha}^2A_iE\bar{c}$. One copy of $\bar{\alpha}$ is sent to the skin membrane, one remains here. Using the rules $\bar{c}A_i \to c\bar{\beta}^2$ and $\bar{\alpha} \to \#$, we pass to $z'' = \#\bar{\beta}^2Ec$. One copy of each symbol from $\bar{\beta}$ is sent to the skin membrane, one remains in membrane i and at the next step is replaced by $\#$.

   Note that if we do not use the rule $cX_i \to \bar{c}\bar{\alpha}^2$ at the first step, then we have to use the rule $cA_i \to \bar{c}\dagger$; similarly, if we do not use the rule $\bar{c}A_i \to c\bar{\beta}^2$ at the second step, then we have to use the rule $\bar{c}E \to c\dagger$. In both cases, the computation will never halt.

   Simultaneously, in the skin membrane we proceed as follows. After the first step we here have a multiset $z_1 = X_iA_iDE'\bar{c}u$, where u consists of all symbols from N_2 which remain here (plus dummy symbols). At the same time, we receive from membrane i a symbol $\bar{Y}$, for some $Y \in N_1$ (or nothing, if the matrix m_i was a terminal one, $m_i : (X \to \lambda, A \to x)$), and E' is replaced by E''. At the next step, the symbols X_i, A_i are replaced by $\#$, while $\bar{c}\bar{Y}$ are replaced by cY and E'' is replaced by E. The multiset still contains the symbols E and $\bar{c}$. At the same time, we receive from membrane i either symbols from N_2 or a symbol $\bar{a}, a \in T$. The multiset contains again D, E, c, a symbol from N_1 and some symbols from N_2, hence we return to a contents as that from the beginning. At the next step, for the symbol $\bar{a}, a \in T$, we produce $n+2$ copies of a; exactly one of these copies enters each of the regions. In all membranes of the system, a is immediately replaced by $\#$. The copy which leaves the system contributes to the result of the current computation.

   Therefore, after three steps in membrane i and three steps in the skin membrane, we have accomplished the simulation of the matrix m_i. The procedure can be iterated.

2. $z_i = X_iA_jEc$, for some $i \neq j$. At the first step we can again use the rule $cX_i \to \bar{c}\bar{\alpha}^2$, simultaneously with $A_j \to \#$, but at the next step we have to introduce the trap-object $\dagger$, because we have to use the rule $\bar{c}E \to c\dagger$. The computation never stops.

3. $z_i = X_iEc$; exactly as in case 2, without using the rule $A_j \to \#$.

4. $z_i = X_j A_i Ec$, for some $j \neq i$. At the first step we have to use the rule $cA_i \to \bar{c}\dagger$ (simultaneously with $X_j \to \#$), so the computation never stops.

5. $z_i = A_i Ec$; exactly as in case 4 (without using the rule $X_j \to \#$).

6. $z_i = X_j A_k Ec$, for some $j \neq i, k \neq i$. We replace X_j, A_k by $\#$ and, from the point of view of membrane i, the computation can continue.

7. $z_i = X_j Ec$, for some $j \neq i$; exactly as in case 6.

8. $z_i = A_j Ec$, for some $j \neq i$; exactly as in case 6.

Therefore, if a symbol X_i or a symbol A_i is introduced, then the computation can correctly continue and halt if and only if both symbols X_i and A_i were introduced. At the first sight, cases 6, 7, 8 do not correctly control the computation, but this is still done: when a symbol X_j or A_k is introduced, $n+2$ copies of it are introduced; one of these copies will reach membrane j, respectively k, and these membranes check whether or not both these symbols are present and whether or not $j = k$.

The derivation in the grammar G stops by using a matrix of the form $m_i = (X \to \lambda,\ A \to \alpha)$, for some $\alpha \in T \cup \{\lambda\}$. When this matrix is simulated in Π, no symbol $\bar{Y}$ is sent from membrane i to the skin membrane. Instead of the rule $\bar{c}\bar{Y} \to cY$, we can use the rule $\bar{c}E'' \to c\bar{E}$. If no symbol from N_2 is present, then the computation stops.

As long as any nonterminal is present in the skin membrane, the computation must continue. If we have only symbols from N_1 or only symbols from N_2, then the computation will never stop (see again cases 3, 5, 7, 8 discussed above). Consequently, a computation stops if and only if it correctly simulates a terminal derivation with respect to G.

From the previous construction, it is now clear that $\Psi_T(L(G)) = Ps(\Pi)$, that is, $PsMAT \subseteq PsLP_*(con, 2Cat_1)$.

In order to prove the strictness of the inclusion we consider the P system

$$\Pi = (\{A, B, D, X, X', X'', Y, c, a, \dagger\}, \{a\}, \{a\}, \{c\}, [_1\ ]_1, cAB, R_1)$$

with

$$\begin{aligned} R_1 \ = \ & \{cA \to \bar{c}AB,\ \bar{c}A \to cA,\ A \to \dagger,\ \bar{c}A \to cXY,\ B \to Baa, \\ & a \to D,\ cB \to \bar{c}D,\ \bar{c}X' \to cX,\ X \to X', \\ & X' \to \dagger,\ \bar{c}X' \to cX'',\ cY \to \bar{c}\dagger,\ \dagger \to \dagger\}. \end{aligned}$$

We start by having one copy of the object B. The first phase of a computation consists in using $n-1, n \geq 1$ times the couple of rules $cA \to \bar{c}AB$, $\bar{c}A \to cA$, which introduces n copies of B. When using the rule $cA \to \bar{c}AB$, in parallel, we also use the rule $B \to Baa$, while in parallel with using the rule $\bar{c}A \to cA$, we use both rules $B \to Baa$ and $a \to D$. This means that for each B, one of the two copies of a

introduced by the rule $B \to Baa$ is sent out of the system and one remains inside; the copy which remains in the system is then replaced by the "dummy" object D. After these $2(n-1)$ steps, we use the rule $cA \to \overline{c}AB, \overline{c}A \to cXY$ – in parallel with $B \to Baa$ and $a \to D$. In this way, we send out of the system 1, 2, 2, 3, 3,..., $n, n, n+1$ copies of a, which is, in total $(n+1)(n+1)-1$, for some $n \geq 1$.

Note that during this phase, we cannot use the catalyst together with another object, because of the rule $A \to \dagger$: as long as A is present, it has to evolve by one of the rules $cA \to \overline{c}AB$, $\overline{c}A \to cA$, otherwise the computation will never terminate.

After removing the object A (and introducing the objects X, Y), we have to use the rule $cB \to \overline{c}B$; one copy of B is replaced by D, the others introduce two copies of a, while X becomes X'. The rule $cB \to \overline{c}D$ must me used, otherwise we have to use $cY \to \overline{c}\dagger$. For each B, one copy of a is sent out, the other one will remain inside the system and it will be replaced by D at the next step. The catalyst returns to the non-barred version by using the rule $\overline{c}X' \to cX$. The process is iterated, and it is somewhat symmetric to the first phase: the number of copies of B is decreased step by step and, in parallel, copies of a are sent out of the system. Again, we have pairs of steps, hence we send out $n+1, n, n, \ldots, 2, 2, 1, 1$, which means in total $(n+1)(n+1)$.

When exactly one copy of B is present, we can use the rule $\overline{c}X' \to cX''$, which can be followed only by $cB \to \overline{c}D$, and then no rule can be used.

Therefore

$$Ps(\Pi) = \{2n^2 + 4n + 1 \mid n \geq 1\}.$$

By the above mentioned fact that all languages $L \in MAT$ where $L \subseteq \{a\}^*$ for some letter a are regular, we obtain $Ps(\Pi) \notin PsMAT$. □

In the proof of the strictness of the inclusion we have used a P system with one region and one catalyst. If we keep the restriction concerning the number of regions but delete that concerning the number of catalysts, we are able to generate all recursively enumerable languages.

Theorem 2. $PsRE = PsLP_1(con, 2Cat_*)$.

Proof. Because $PsMAT_{ac} = PsRE$ and because the inclusion $PsLP_1(con, 2Cat_*) \subseteq PsRE$ is straightforward, we only have to prove that $PsMAT_{ac} \subseteq PsLP_1(con, 2Cat_*)$.

Starting from a matrix grammar $G = (N, T, S, M, F)$ in strong binary normal form with $N = N_1 \cup N_2 \cup \{S, \dagger\}$ and n matrices of the form $m_i : (X \to Y, A \to \alpha)$, $1 \leq i \leq n$, and k matrices of the form $m_{n+j} : (X \to Y, A \to \dagger)$, $1 \leq j \leq k$, we construct the corresponding P system

$$\Pi = (V, T, C, [_1]_1, w_1, R_1)$$

where

$$\begin{aligned} V &= N \cup T \cup C \cup \{E, \dagger\} \cup \{X', X'' \mid X \in N_1\}, \\ C &= \{c_i, \overline{c_i} \mid 1 \leq i \leq n\} \cup \{d_j, \overline{d_j} \mid 1 \leq j \leq k\}, \\ w_1 &= XAEc_1c_2 \ldots c_n d_1 d_2 \ldots d_k, \end{aligned}$$

where $(S \to XA)$ is the start matrix in M, and the set R_1 contains the following rules:

1. For each matrix $m_i : (X \to Y,\ A \to \alpha)$, $1 \leq i \leq n$, we consider the rules:
$$c_i X \to \overline{c_i} Y',\ Y' \to Y,\ \overline{c_i} E \to c_i \dagger,\ \overline{c_i} A \to \overline{\alpha},$$
where $\overline{\alpha} = \alpha^2$ for $\alpha \in T$, and $\overline{\alpha} = \alpha$ otherwise.
2. For each matrix $m_{n+j} : (X \to Y,\ A \to \dagger)$, $1 \leq j \leq k$, we consider the rules:
$$d_j X \to \overline{d_j} Y',\ Y' \to Y'',\ \overline{d_j} A \to d_j \dagger,\ \overline{d_j} Y'' \to d_j Y .$$
3. For the final matrix $(Z \to \lambda)$ we take the rule $Z \to \lambda$, and we also use the rule $\dagger \to \dagger$ to ensure that the computation never stops if the trap symbol $\dagger$ has been introduced.

At any moment, except after the last step of a computation, where the final rule $Z \to \lambda$ is applied, exactly one symbol from $N_1 \cup N_1' \cup N_1''$ is present. If a matrix $m_i : (X \to Y,\ A \to \alpha)$, $1 \leq i \leq n$, is simulated, then first the first production $X \to Y$ is evolved together with the catalyst c_i yielding Y from X in two steps, and in the second step the corresponding second rule $A \to \alpha$ must be simulated together with the catalyst $\overline{c_i}$, otherwise the trap symbol $\dagger$ is introduced. For $\alpha \in T$ we generate a second copy, which can leave the system.

For simulating a matrix $m_{n+j} : (X \to Y,\ A \to \dagger)$, $1 \leq j \leq k$, three computation steps in Π are necessary: if the symbol A is present, in the second step catalyst $\overline{d_j}$ must react together with A thus generating the trap symbol $\dagger$; otherwise, $\overline{d_j}$ can wait for being evolved to d_j again together with Y''.

A computation in Π now stops if and only if the final matrix has been simulated and no trap symbol $\dagger$ is present, i.e., exactly if and only if a successful derivation in G has been simulated correctly in Π.

Consequently, we obtain $\Psi_T(L(G)) = \Psi_T(L(\Pi))$, which concludes the proof. □

We close this paper with the observation that the concentration of symbols was already used in [1] as a control mechanism of L systems. We can combine the idea of [1] with that proposed in this paper and we can consider a sort of "bi-cameral DOL systems", as a pair of DOL systems (morphisms) which work separately on multisets of symbols and, after each rewriting step, redistribute the symbols in the same way as in a P system with a concentration-controlled communication. As the output of such a device we consider the union of the two multisets (or the Parikh image of this union). The power and the properties of these cooperative DOL systems remain to be investigated. Anyway, it is clear that they can induce growth functions and length sets which are not growth functions or length sets of usual DOL systems: take two different unary DOL systems; by redistribution we will get a length set which does not consist of the powers of a given natural number, as the length set of a unary DOL language is.

Acknowkedgement. The authors are very grateful to the referee for her/his detailed comments and proposals for an improvement. Especially, Lemma 1 and its use in the proof of Theorem 2 belong to the referee.

References

[1] J. Dassow, Concentration dependent 0L systems, *Intern. J. Computer Math.*, 7 (1979), 187–206.

[2] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[3] J. Dassow, Gh. Păun, On the power of membrane computing, *J. of Universal Computer Sci.*, 5, 2 (1999), 33–49 (www.iicm.edu/jucs).

[4] R. Freund, Generalized P systems, *Fundamentals of Computation Theory, FCT'99*, Iaşi, 1999, (G. Ciobanu, Gh. Păun, eds.), *LNCS* 1684, Springer, 1999, 281–292.

[5] R. Freund, F. Freund, Molecular computing with generalized homogeneous P systems, *Proc. Conf. DNA6* (A. Condon, G. Rozenberg, eds.), Leiden, 2000, 113–125.

[6] D. Hauschildt, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Inform.*, 31 (1994), 719–728.

[7] S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), 357–367.

[8] W. R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford Univ. Press, New York, Oxford, 1999.

[9] S. S. Mader, *Biology* (Fifth Edition), McGraw-Hill, Boston, 1996 (Chapter 6: *Membrane structure and function*, 84–102).

[10] C. Martin-Vide, V. Mitrana, P systems with valuations, in vol. *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 154–166.

[11] Gh. Păun, Computing with membranes, *J. Computer and System Sci.*, 61, 1 (2000), 108–143 (see also *TUCS Research Report* No. 208, November 1998, http://www.tucs.fi).

[12] Gh. Păun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (Febr. 1999), 139–152.

[13] Gh. Păun, Computing with membranes. A variant, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.

[14] Gh. Păun, P systems with active membranes: Attacking NP complete problems, *J. Automata, Languages, and Combinatorics*, 6, 1 (2001), 75–90.

[15] Gh. Păun, Y. Sakakibara, T. Yokomori, P systems on graphs of restricted forms, submitted, 1999.

[16] Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397–410.

[17] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.

[18] Y. Suzuki, H. Tanaka, On a LISP implementation of a class of P systems, *Romanian J. of Information Science and Technology*, 3, 2 (2000), 173–186.

[19] Cl. Zandron, Cl. Ferretti, G. Mauri, Solving NP complete problems using P systems with active membranes, in vol. *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer-Verlag, London, 2000, 289–301.

Received August, 2000

Acta Cybernetica 15 (2001) 25-32.

A Chomsky-Schützenberger-Stanley Type Characterization of the Class of Slender Context-Free Languages*

Pál Dömösi † Satoshi Okawa ‡

Abstract

Slender context-free languages have a complete algebraic characterization by L. Ilie in [13]. In this paper we give another characterization of this class of languages. In particular, using linear Dyck languages instead of unrestricted ones, we obtain a Chomsky-Schützenberger-Stanley type characterization of slender context-free languages.

1 Introduction

We consider slender languages, that is, languages for which the number of words of the same length is bounded by a constant. As proved in [16], the slender regular languages are exactly the disjoint unions of single loops, that is, disjoint finite unions of sets of the form uv^*w. A similar characterization holds for slender context-free languages as disjoint unions of paired loops, that is, finite unions of sets of the form $\{uv^nwx^ny \mid n \geq 0\}$ [13, 17].

The characterization of language classes by algebraic operations is one of the most important issues in formal language theory. Chomsky-Schützenberger-Stanley's characterization [1, 2, 20, 21] for the class of context-free languages was the first well-known result in this direction and is stated as follows: For any context-free language L, there exists a regular language R such that $L = h(R \cap D)$ where D is a Dyck language and h is a homomorphism. Moreover, it is clear that $h(R \cap D)$

*This work has been supported by the joint project "Automata & Formal Languages" of the Hungarian Academy of Sciences and Japanese Society for Promotion of Science (No. 15) "Automata & Formal Languages"and by the Hungarian National Foundation for Scientific Research (OTKA T030140).

†Institute of Mathematics and Informatics, Debrecen University, Debrecen, Egyetem tér 1, H-4032, Hungary, e-mail: domosi@math.klte.hu

‡Faculty of Computer Science and Engineering, the University of Aizu, Aizu-Wakamatsu, 965-8580, Japan, e-mail: okawa@u-aizu.ac.jp

is a context-free language for each regular language R by the closure properties of the class of context-free languages. A refinement of this classical result is shown in [11].

For recursively enumerable languages, a Chomsky-Schützenberger-Stanley type characterization is given in [10]. It is also proved [15] that there exists no characterization of this type for context-sensitive languages. (See some other types of homomorphic characterizations of recursively enumerable languages in [3, 4, 5, 7, 9].)

In this paper we investigate a characterization of Chomsky-Schützenberger-Stanley type for slender context-free languages.

However, a Chomsky-Schützenberger-Stanley type characterization of the class of slender context-free languages is almost meaningless, because a slender context-free language is linear but a Dyck language is not linear. If we use a Dyck language for characterization, then, in fact, we use complex languages to characterize simpler ones. We consider another characterization which is similar to Chomsky-Schützenberger-Stanley's one.

This paper is organized as follows. In Section 2, we introduce some fundamental notions, notations, definitions of slender languages, and the loop characterization results for slender languages. In Section 3, we give our main theorem, which offers a Chomsky - Stanley type characterization of the class of slender context-free languages. Section 4 gives some concluding remarks.

2 Preliminaries

For all notions and notations not defined here, see [6, 8, 12, 14, 18, 19]. By an *alphabet* Σ we mean a finite nonvoid set. An element of Σ is called a *letter*. A *word* over Σ is a finite sequence of elements in Σ. For a word w, the *length* $|w|$ is the number of letters in w, where each letter is counted as many times as it occurs. The set of all the words over Σ is denoted by Σ^*. In particular, λ is a word in Σ^* and is called the *empty word*. Thus $|\lambda| = 0$. If u and v are words over an alphabet Σ, then their *catenation* uv is also a word over Σ. It is clear that λ acts as identity for this operation, that is, for every word u over Σ, $u\lambda = \lambda u = u$. Therefore, Σ^* becomes a free monoid with catenation as the multiplication and λ as the identity, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is a semigroup.

Any subset of Σ^* is referred to as a *language* over Σ.

Now we define slender languages. A language $L \subseteq \Sigma^*$ is said to be *k-slender* if $card\{w \in L \mid |w| = n\} \leq k$ for every $n \geq 0$. A language is *slender* if it is k-slender for some positive integer k. In particular, a 1-slender language is called a *thin* language.

For the loop characterization of slenderness, some notation and definitions are introduced. For a word u, setting $u^0 = \lambda$ and $u^n = u^{n-1}u$ for $n > 0$, we define u^* and u^+ as usual, by $u^* = \{u^n \mid n \geq 0\}$ and $u^+ = u^* \setminus \{\lambda\}$.

A language L is said to be a *union of single loops* (or, in short, USL) if for some

positive integer k and some words $u_i, v_i, w_i, 1 \leq i \leq k$,

$$(*) \quad L = \bigcup_{i=1}^{k} \{u_i\}\{v_i\}^*\{w_i\}.$$

A language L is called a *union of paired loops* (or UPL, in short) if for some positive k and some words $u_i, v_i, w_i, x_i, y_i, 1 \leq i \leq k$,

$$(**) \quad L = \bigcup_{i=1}^{k} \{u_i v_i^n w_i x_i^n y_i \mid n \geq 0\}.$$

A USL language L is said to be a *disjoint union of single loops* (DUSL, in short) if the sets in the union $(*)$ are pairwise disjoint. The notion of a *disjoint union of paired loops* (DUPL) is defined analogously considering $(**)$.

A *grammar* is an ordered quadruple $G = (N, \Sigma, S, P)$ where N and Σ are disjoint alphabets of *variables* and *terminals*, respectively, the *start symbol* $S \in N$, and P is a finite set of ordered pairs (α, β) called *productions*, such that β is a word over the alphabet $N \cup \Sigma$ and α is a word over $N \cup \Sigma$ containing at least one letter of N. Usually, a production is written $\alpha \to \beta$ instead of (α, β).

For a word ξ over $N \cup \Sigma$, if ξ is decomposed into

$$\xi = \xi_1 \alpha \xi_2$$

and $\alpha \to \beta$ is a production in P, then $\alpha \to \beta$ is applicable to ξ and the result of the application is a word $\eta = \xi_1 \beta \xi_2$. We say that ξ *derives directly* η, and write $\xi \Rightarrow \eta$.

The *language generated by a grammar* $G = (N, \Sigma, S, P)$ is the set $L(G) = \{w \mid w \in \Sigma^*$ and $S \Rightarrow^* w\}$, where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

If $\alpha \to \beta \in P$ implies $\alpha \in N$ then G is called *context-free*. A context-free grammar is said to be *linear* if for every production $\alpha \to \beta \in P$, $\beta \in \Sigma^* N \Sigma^* \cup \Sigma^*$ A linear grammar is called *right-linear* or *regular* if for every production $\alpha \to \beta \in P$, $\beta \in \Sigma^* N \cup \Sigma^*$. $L \subseteq \Sigma^*$ is a *regular (linear, context-free) language* if we have $L = L(G)$ for some regular (linear, context-free) grammar G.

Let $G = (N, \Sigma, S, P)$ be a context-free grammar with $N = \{S\}$, $\Sigma = \{a_i, a_i' \mid i = 1, \ldots, n\}$, and $P = \{S \to \lambda, S \to SS\} \cup \{S \to a_i S a_i', \mid i = 1, \ldots, n\}$. We say that G and $L(G)$ are a *Dyck grammar* and the *Dyck language over the 2n-letter alphabet* Σ, respectively. Furthermore, if the set of productions of a grammar $G_{\mathcal{L}}$ is $P_{\mathcal{L}} = \{S \to \lambda\} \cup \{S \to a_i S a_i', \mid i = 1, \ldots, n\}$, then $G_{\mathcal{L}}$ is called a *linear Dyck grammar* and its language $L(G_{\mathcal{L}})$ is called a *linear Dyck language*.

We shall use the following well-known results about slender languages.

Theorem 2.1. [16] *The following conditions are equivalent for a language L:*

(i) L is regular and slender.

(ii) L is USL.

(iii) L is DUSL. □

Theorem 2.2. [16] *Every UPL language is DUPL, slender, linear and unambiguous.* □

Theorem 2.3.[13, 17] *Every slender context-free language is UPL.* □

We have the following direct consequence of Theorems 2.2 and 2.3:

Proposition 2.4. *The class of slender linear languages coincides with the class of slender context-fee languages. In addition, the class of slender context-free languages contains only unambiguous languages.* □

3 Results

As stated in the Introduction, a Chomsky-Schützenberger-Stanley type characterization of the class of slender context-free languages is almost meaningless, because a slender context-free language is linear but a Dyck language is not linear. Therefore, we use linear Dyck languages instead of Dyck languages in our Chomsky-Stanley type characterization.

Theorem 3.1. *Let Σ be an alphabet. Then a Δ, a homomorphism $h : \Delta^* \to \Sigma^*$ and a linear Dyck language $D_{\mathcal{L}}$ on Δ can be determined from Σ, such that for every slender context-free language $L \subseteq \Sigma^*$ there can be found a regular language $R \subseteq \Delta^*$ with $L = h(R \cap D_{\mathcal{L}})$.*

Proof. Let Σ be an alphabet. Then we first define an alphabet Δ, a homomorphism h, and the linear Dyck language $D_{\mathcal{L}}$ on Δ as follows:

An alphabet Δ is defined by

$$\Delta = \Sigma \cup \Sigma' \cup \bar{\Sigma} \cup \bar{\Sigma}',$$

where

$$\Sigma' = \{a' \mid a \in \Sigma\}, \bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}, \text{ and } \bar{\Sigma}' = \{\bar{a}' \mid a \in \Sigma\}.$$

The homomorphism $h : \Delta^* \to \Sigma^*$ is defined by

$$h(a) = h(\bar{a}') = a,\ a \in \Sigma \text{ and } h(x) = \lambda,\ x \in \Delta \setminus (\Sigma \cup \bar{\Sigma}').$$

The linear Dyck language $D_{\mathcal{L}}$ over Δ is the language generated by

$$G_{\mathcal{L}} = (\{S\}, \Delta, S, P_{\mathcal{L}}),$$

where

$$P_{\mathcal{L}} = \{S \to aSa', S \to \lambda \mid a \in \Sigma \cup \bar{\Sigma}\}.$$

In order to simplify the notations, we use the following abbreviations. For a word $w = a_1 \dots a_\ell \in \Sigma^*$, we have $w^R = a_\ell \dots a_2 a_1$, $w' = a'_1 \dots a'_\ell, \bar{w} = \bar{a}_1 \dots \bar{a}_\ell$, and $\bar{w}' = \bar{a}'_1 \dots \bar{a}'_\ell$.

Let L be any slender context-free language over Σ. By Theorem 2.3 we can find a finite index set I and words u_i, v_i, w_i, x_i, y_i, for all $i \in I$ with $L = \bigcup_{i \in I} \{u_i v_i^n w_i x_i^n y_i \mid n \geq 0\}$.

Define a regular grammar $G_L = (N, \Delta, A, P_R)$, where $N = \{A\} \cup \{B_i, C_i \mid i \in I\}$, $P_R = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5$ as

$P_1 = \{A \to u_i \bar{y_i}^R B_i \mid i \in I\}$,

$P_2 = \{B_i \to v_i \bar{x_i}^R B_i \mid i \in I\}$,

$P_3 = \{B_i \to w_i w_i'^R C_i \mid i \in I\}$,

$P_4 = \{C_i \to \bar{x_i}' v_i'^R C_i \mid i \in I\}$, and

$P_5 = \{C_i \to \bar{y_i}' u_i'^R \mid i \in I\}$.

Let R be a language generated by G_L, i.e., $R = L(G_L)$. Then $L = h(R \cap D_{\mathcal{L}})$ can be proved by

a.) $L \subset h(R \cap D_{\mathcal{L}})$.

Suppose w is in L and w is of the form $u_i v_i^n w_i x_i^n y_i$ for some i and n.

By the definition of G_L, it is clear that a word

$$\xi = u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R (\bar{x_i}' v_i'^R)^n \bar{y_i}' u_i'^R$$

is generated by G_L as follows:

$A \Rightarrow u_i \bar{y_i}^R B_i \Rightarrow u_i \bar{y_i}^R v_i \bar{x_i}^R B_i \Rightarrow^* u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n B_i \Rightarrow u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R C_i$
$\Rightarrow u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R \bar{x_i}' v_i'^R C_i \Rightarrow^* u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R (\bar{x_i}' v_i'^R)^n C_i$
$\Rightarrow^* u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R (\bar{x_i}' v_i'^R)^n \bar{y_i}' u_i'^R$.

Moreover, it is clear that ξ is in $D_{\mathcal{L}}$, and therefore ξ is in $R \cap D_{\mathcal{L}}$. By the definition of h, $h(\xi)$ is a word $u_i v_i^n w_i x_i^n y_i$, i.e., w. So w is a word in $h(R \cap D_{\mathcal{L}})$.

b.) $h(R \cap D_{\mathcal{L}}) \subset L$.

Let $w \in h(R \cap D_{\mathcal{L}})$. Then, there is a word ξ in $R \cap D_{\mathcal{L}}$ such that $w = h(\xi)$. By the definition of G_L, ξ should be of the form

$$\xi = u_i \bar{y_i}^R (v_i \bar{x_i}^R)^m w_i w_i'^R (\bar{x_i}' v_i'^R)^n \bar{y_i}' u_i'^R$$

for some $i \in I$. By the definition of $D_{\mathcal{L}}$, $n = m$. Hence, $\xi = u_i \bar{y_i}^R (v_i \bar{x_i}^R)^n w_i w_i'^R (\bar{x_i}' v_i'^R)^n \bar{y_i}' u_i'^R$ and $h(\xi) = u_i v_i^n w_i x_i^n y_i$. Therefore, $w = h(\xi)$ is in L.

This completes the proof. □

Remark. There exists a regular language R such that $h(R \cap D_{\mathcal{L}})$ is not slender.

For example, choose a regular language Δ^* as R. Then, by the fact that $R \cap D_{\mathcal{L}}$ is $D_{\mathcal{L}}$ and the fact that $h(D_{\mathcal{L}})$ is Σ^*, the remark follows. Because of the previous observation, it is interesting to find a class $\mathcal{C}$ of regular languages satisfying the following condition: For any slender context-free language L, we can find R in $\mathcal{C}$ such that $L = h(R \cap D_{\mathcal{L}})$, and for any R in $\mathcal{C}$, $h(R \cap D_{\mathcal{L}})$ is a slender context-free language.

We denote by $\mathcal{R}_D$ the class of regular languages that satisfy the condition mentioned above.

A language L is called a *union of double loops* (or UDL, in short) if for words u_i, v_i, w_i, x_i, y_i where $1 \leq i \leq k$,

$$L = \bigcup_{i=1}^{k} \{u_i v_i^* w_i x_i^* y_i\}.$$

It is clear that L is regular by the definition. However, it is clear by Theorem 2.1 that L is not slender.

Now we have the following result, a little stronger than Theorem 3.1:

Theorem 3.2. *Let Σ be an alphabet. Then an alphabet Δ, a homomorphism $h : \Delta^* \to \Sigma^*$ and a linear Dyck language $D_{\mathcal{L}}$ on Δ can be determined from Σ such that for every slender context-free language $L \subseteq \Sigma^*$, there can be found a UDL regular language $R \subseteq \Delta^*$ such that $L = h(R \cap D_{\mathcal{L}})$. Moreover, for any UDL regular language R, $h(R \cap D_{\mathcal{L}})$ is slender context-free.*

Proof. In the proof of Theorem 3.1, one can see the fact that R is a UDL regular language, so the first part of the theorem holds.

Now we consider the second part. Let R be a UDL regular language. Then, since the class of linear context-free languages is closed under the operation of intersection with a regular set, $R \cap D_{\mathcal{L}}$ is linear. Furthermore, by counting the number of words of length n in $R \cap D_{\mathcal{L}}$, we can find that $R \cap D_{\mathcal{L}}$ is slender. Indeed, by the simmetricity of the elements of $D_{\mathcal{L}}$, every $uv^\ell wx^m y \in D_{\mathcal{L}}$ has the form $a_1 \dots a_f (a_{f+1} \dots a_g)^\ell a_{g+1} \dots a_h a'_h \dots a'_{g+1} (a'_g \dots a'_{f+1})^m a'_f \dots a'_1$ with $k = \ell$ and $|v| = |x|$. Hence, by $L = \cup_{i=1}^{k} \{u_i v_i^* w_i x_i^* y_i\}$, the language $R \cap D_{\mathcal{L}}$ has at most k words of length n for every $n \geq 1$. Since the class of slender context-free languages is closed under homomorphisms, $h(R \cap D_{\mathcal{L}})$ is slender context-free.

This completes the proof. □

4 Concluding Remarks

In this paper, we investigated some Chomsky-Schützenberger-Stanley type homomorphic characterizations for slender context-free languages and obtained the first characterization as Theorem 3.1 and the second characterization as Theorem 3.2, by which any slender language can be represented by the homomorphic image of the intersection of a linear Dyck language and a UDL regular language, and for any UDL regular language, the homomorphic image of its intersection with a linear Dyck language is slender. This means that the second result is stronger than the first one.

References

[1] Chomsky, N., Context-free grammars and pushdown storage, *M.I.T. Res. Lab., Electron Quart. Prog. Rept.*, **65** (1962) 187-194.

[2] Chomsky, N., Schützenberger, M., The algebraic theory of context-free languages, *Comput. Programming and Formal Systems, North-Holland, Amsterdam* (1963), 118–161.

[3] Culik, K. II, A purely homomorphic characterization of recursively enumerable sets, *J. ACM* **26** (1979) 345-350.

[4] Engelfriet, J., Rozenberg, G., Fixed point languages, equality languages and representation of recursively enumerable languages, *J. ACM* **27** (1980) 499-518.

[5] Fülöp, Z., Vágvölgyi, S., On ranges of compositions of deterministic root-to-frontier tree transformations, *Acta Cybernet.* **8** (1988), 259-266.

[6] Ginsburg, S., The Mathematical Theory of Context-Free Languages, *McGraw-Hill Book Company, New York, St Louis, San Francisco, Auckland, Bogota, Hamburg, Johannesburg, London, Madrid, Mexico, Montreal, New Delhi, Panama, Paris, Sao Paulo, Singapore, Sydney, Tokyo, Toronto,* 1966.

[7] Ginsburg, S., Greibach, S. A., Harrison, M. A., One-way stack automata, *J. ACM* **14** (1967) 389/418.

[8] Harrison, M. A., Introduction to Formal Language Theory, *Addison-Wesley Publishing Company, Reading , Massachusetts, Menlo Park, California, London, Amsterdam, Don Mils, Ontario, Sidney,* 1978.

[9] Hirose, S., Nasu, M., Left universal context-free grammars and homomorphic characterizations of languages, *Inform. and Control,* **50** (1981) 110-118.

[10] Hirose, S., Okawa, S., Yoneda, M., A homomorphic characterization of recursively enumerable languages, *Theoret. Comput.Sci.* **35** (1985) 261-269.

[11] Hirose, S., Yoneda, M., On the Chomsky's and Stanley's homomorphic characterization of context-free languages, *Theoret. Comput.Sci.* **36** (1985) 109-112.

[12] Hopcroft, J. E. & Ullmann, J. D., Introduction to Automata Theory, Languages, and Computation , *Addison-Wesley, Reading , Massachusetts, Menlo Park, California, London, Amsterdam, Don Mils, Ontario, Sidney,* 1979.

[13] Ilie, L., On a conjecture about slender context-free languages, Theoret. Comput.Sci. **132** (1994) 427-434.

[14] Imreh, B., Ito, M., A note on the regular strongly shuffle-closed languages, *Acta Cybernet.* **12** (1995), 11-22.

[15] Okawa, S., Hirose, S., Yoneda, M., On the impossibility of the homomorphic characterization of context-sensitive languages, *Theoret. Comput.Sci.* **44** (1986) 225-228.

[16] Păun, G., Salomaa, A., Thin and slender languages, *Discrete Appl. Math.*, **61** (1995) 257-270.

[17] Raz, D., Length considerations in context-free languages, *Theoret. Comput.Sci.* **183** (1997) 21-32.

[18] Révész, Gy. E., Introduction to Formal Languages, *McGraw-Hill, New York, St Louis, San Francisco, Auckland, Bogota, Hamburg, Johannesburg, London, Madrid, Mexico, Montreal, New Delhi, Panama, Paris, Sao Paulo, Singapore, Sydney, Tokyo, Toronto,* 1983.

[19] Salomaa, A., Formal Languages, *Academic Press, New York, London,* 1973.

[20] Schützenberger, M., On context-free languages and push-down automata, *Inf. Control* **6** (1963), 246–264.

[21] Stanley, R. J., Finite state representations of context-free languages, *M.I.T. Res. Lab., Electron Quart. Prog. Rept.*, **76** (1965) 276-279.

Received November, 2000

Acta Cybernetica 15 (2001) 33-44.

On isomorphic representations of generalized definite automata*

Ferenc Gécseg † Balázs Imreh†

To Professor Magnus Steinby on his 60th birthday

Abstract

In this paper, the generalized definite automata are studied. In particular, systems which are isomorphically complete for this class with respect to the α_i-products are characterized.

1 Introduction

Generalized definite languages and recognizers were introduced by Ginzburg in [5]. Generalized definite automata were also studied in the works [2], [7], [8]. This class is so wide that the classes of definite and reverse definite automata are its proper subclasses (see [7]). Here, we deal with the isomorphic representations of the generalized definite automata with respect to the α_i-products. In particular, necessary and sufficient conditions are given for systems of automata to be isomorphically complete for this class with respect to the α_i-products. The paper is organized as follows. After the preliminaries of Section 2, we recall the characterizations of the subdirectly irreducible definite, reverse definite, and generalized definite automata in Section 3. Then we describe the isomorphically complete systems for the class of all generalized definite automata with respect to the α_i-products.

2 Preliminaries

In what follows, X always denotes a finite alphabet, and as usual X^* denotes the set of all words over X. For every nonnegative integer j, let $X^j = \{w : w \in X^* \text{ and } |w| = j\}$, where $|w|$ denotes the length of the word w.

By an *automaton* we mean a pair $\mathbf{A} = (A, X)$, where A is a finite nonempty set of *states*, X is a finite nonempty set of the *input symbols*, and every $x \in X$ is realized as a unary operation $x^{\mathbf{A}} : A \to A$. For any word $w = x_1 \dots x_s \in X^*$,

*his work has been supported by the the Hungarian National Foundation for Scientific Research, Grant T030143, and the Ministry of Culture and Education of Hungary, Grant FKFP 0704/1997.

†Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary

$w^{\mathbf{A}} : A \to A$ is defined as the composition of the mappings $x_1^{\mathbf{A}}, \ldots, x_s^{\mathbf{A}}$. If $\mathbf{A}$ is known from the context, we write simply aw for $aw^{\mathbf{A}}$.

By the definition above, an automaton can be considered as a unoid. Therefore, such notions as subautomata, congruences, homomorphisms, isomorphisms, embeddings, direct products, subdirect products, subdirect irreducibility can be defined in the usual way (see *e.g.* [1] or [6]). We shall use a particular isomorphism defined as follows. Let $\mathbf{A} = (A, X)$ and $\mathbf{B} = (B, Y)$ be two automata, μ a one-to-one mapping of A onto B and τ a one-to-one mapping of X onto Y. Then the pair μ, τ of mappings is called an (A, X)-*isomorphism* of $\mathbf{A}$ onto $\mathbf{B}$ if $ax^{\mathbf{A}}\mu = a\mu(x\tau)^{\mathbf{B}}$ is valid, for all $a \in A$ and $x \in X$. In this case, it is said that $\mathbf{A}$ and $\mathbf{B}$ are (A, X)-*isomorphic*.

We introduce some particular congruence relations which we need in the sequal. For this purpose, let $\mathbf{A} = (A, X)$ be an arbitrary automaton with at least three states. A congruence ρ of $\mathbf{A}$ is called *elementary* if $\rho = \omega_A \cup \{(a, b), (b, a)\}$ for two distinct states $a, b \in A$, where ω_A denotes the *diagonal relation*, *i.e.*, $\omega_A = \{(a, a) : a \in A\}$. Let us denote by $Con_e(\mathbf{A})$ the set of all elementary congruences of $\mathbf{A}$.

Now, let $j \geq 0$ be an arbitrary integer. Let us define the relation ρ_j on A by

$$a\rho_j b \text{ if and only if } ap = bp, \text{ for all } p \in X^j.$$

It is easy to see that for every nonnegative integer j, ρ_j is a congruence relation of $\mathbf{A}$.

For every integer $j \geq 0$, let us define the state set A_j as follows:

$$A_0 = \{a : a \in A \text{ and } ax = a, \text{ for all } x \in X\},$$

$$A_{j+1} = \{a : a \in A \text{ and } ax \in A_j, \text{ for all } x \in X\}.$$

Then $\mathbf{A}_j = (A_j, X)$ is a subautomaton of $\mathbf{A}$ provided that $A_0 \neq \emptyset$, and the Rees congruences defined by

$$a\sigma_j b \text{ if and only if } a, b \in A_j \text{ or } a = b$$

are congruence relations of $\mathbf{A}$.

For any integer $k \geq 0$, an automaton $\mathbf{A} = (A, X)$ is called *weakly k-definite*, if $|Ap| = 1$, for every $p \in X^k$. Moreover, it is said that $\mathbf{A}$ is *definite* if it is weakly k-definite for some integer $k \geq 0$. In particular, if a weakly k-definite automaton $\mathbf{A}$ has such a state a^*, called *dead state*, that $a^*x = a^*$, for all $x \in X$, then $\mathbf{A}$ is called a *nilpotent automaton*. In this case $Ap = \{a^*\}$ holds, for every $p \in X^k$.

For any integer $k \geq 0$, an automaton $\mathbf{A} = (A, X)$ is called *weakly reverse k-definite* if $apx = ap$ is valid, for all $a \in A$, $p \in X^k$, and $x \in X$. $\mathbf{A}$ is *reverse definite* if it is weakly reverse k-definite for some $k \geq 0$.

Following [8], for any pair of integers $h, k \geq 0$, an automaton $\mathbf{A} = (A, X)$ is called *weakly (h, k)-definite* if $aupv = auv$ is valid, for all $a \in A$, $u \in X^h$, $v \in X^k$, and $p \in X^*$. It is worth noting that for every pair of integers $h' \geq h$, $k' \geq k$, an

automaton $\mathbf{A}$ is weakly (h', k')-definite if it is weakly (h, k)-definite. An automaton is called *generalized definite* if it is weakly (h, k)-definite for some integers $h, k \geq 0$. Let us denote by $\mathcal{G}$ the class of all generalized definite automata. By the definitions, one can obtain the following observation.

Lemma 1. *If* $\mathbf{A} \in \mathcal{G}$ *and* $\mathbf{B}$ *is a homomorphic image of* $\mathbf{A}$*, then* $\mathbf{B} \in \mathcal{G}$ *as well.*

We recall here the notion of α_i-products (see *e.g.* [3], [4]). This product family is a natural generalization of the serial connection or cascade product of automata.

Let i be an arbitrary nonnegative integer. Let us consider the automata $\mathbf{A} = (X, A)$, $\mathbf{A}_j = (X_j, A_j)$, $j = 1, \ldots, m$, and let Φ be a family of feedback functions below

$$\varphi_j : A_1 \times \ldots \times A_{j+i-1} \times X \to X_j, \quad j = 1, \ldots, m.$$

It is said that $\mathbf{A}$ is the *α_i-product* of $\mathbf{A}_j$, $j = 1, \ldots, m$, if the following conditions are satisfied:

(1) $A = \prod_{j=1}^{m} A_j$,

(2) for all $(a_1, \ldots, a_m) \in A$ and $x \in X$,

$$(a_1, \ldots, a_m)x^{\mathbf{A}} = (a_1 x_1^{\mathbf{A}_1}, \ldots, a_m x_m^{\mathbf{A}_m})$$

is valid where $x_j = \varphi_j(a_1, \ldots, a_{j+i-1}, x)$, for all $j \in \{1, \ldots, m\}$.

For the α_i-product introduced above, we use the notation

$$\mathbf{A} = \prod_{j=1}^{m} \mathbf{A}_j(X, \Phi) .$$

When the component automata $\mathbf{A}_j$ are equal, say $\mathbf{A}_j = \mathbf{B}$, $j = 1, \ldots, m$, then it is said that the α_i-product $\mathbf{A}$ is an *α_i-power* of $\mathbf{B}$ and it is denoted by $\mathbf{B}^m(X, \Phi)$.

In particular, if each of the feedback functions is independent of the states, *i.e.*, if the feedback functions have the forms $\varphi_j : X \to X_j$, $j = 1, \ldots, m$, then the α_i-product is called *quasi-direct product.* It corresponds to the parallel connection of automata where the sign transformation is allowed.

Lemma 2. *If an automaton* $\mathbf{A}$ *can be embedded into an* α_0*-product of automata* $\mathbf{A}_j$, $j = 1, \ldots, k$, *moreover, each automaton* $\mathbf{A}_j$ *can be embedded into an* α_0*-product of automata* $\mathbf{A}_{jt}$, $t = 1, \ldots, m_j$, *then* $\mathbf{A}$ *can be embedded into an* α_0*-product of automata* $\mathbf{A}_{jt}$, $t = 1, \ldots, m_j$; $j = 1, \ldots, k$.

Let $\mathcal{N}$ be an arbitrary class and $\mathcal{M}$ a system of automata. It is said that $\mathcal{M}$ is *isomorphically complete* for $\mathcal{N}$ with respect to the α_i-product if for any automaton $\mathbf{B} \in \mathcal{N}$, there exist automata $\mathbf{A}_j \in \mathcal{M}$, $j = 1, \ldots, m$, such that $\mathbf{B}$ can be embedded into an α_i-product of the automata $\mathbf{A}_j$, $j = 1, \ldots, m$.

3 Isomorphic representations

We shall use the subdirectly irreducible definite, reverse definite, and generalized definite automata, whose charcterizations can be found in [2].

Proposition 1 ([2]). *A definite automaton* $\mathbf{A}$ *with* $|A| \geq 3$ *is subdirectly irreducible if and only if* $Con_e(\mathbf{A}) = \{\rho_1\}$.

To characterize the subdirectly irreducible reverse definite automata we need some preparations.

For any $m \geq 2$, let $X_m = \{x_1, \ldots, x_m\}$ and define the sets $A(m,k)$, $k = 0, 1, \ldots$, inductively so that

$A(m,0) = \{0,1\}$,

$A(m,1) = \{0,1\} \cup \{(i_1, \ldots, i_m) \in \{0,1\}^m : \ i_r \neq i_s \text{ for some } 1 \leq r < s \leq m\}$,

and for any $k \geq 1$,

$A(m,k+1) =$

$A(m,k) \cup \{(i_1, \ldots, i_m) \in A(m,k)^m : \{i_1, \ldots, i_m\} \cap (A(m,k) \setminus A(m,k-1)) \neq \emptyset\}$.

For any $m \geq 2$ and $k \geq 1$, we define an automaton $\mathbf{A}(m,k) = (A(m,k), X_m)$ as follows:

(1) for both $i \in A(m,0)$ and $x \in X_m$, let $ix = i$;

(2) for all $(i_1, \ldots, i_m) \in A(m,k) \setminus A(m,0)$ and $x_s \in X_m$, let $(i_1, \ldots, i_m)x_s = i_s$.

It is clear that for any $m \geq 2$ and $k \geq 1$, $\mathbf{A}(m,k)$ is a subautomaton of $\mathbf{A}(m,k+1)$.

We recall that an automaton $\mathbf{A} = (A, X)$ is *input reduced* if $x^{\mathbf{A}} \neq y^{\mathbf{A}}$ for all pairs of distinct input symbols $x, y \in X$.

Proposition 2 ([2]). *Let* $\mathbf{A} = (A, X)$ *be an input reduced automaton such that* $|A| \geq 3$ *and* $|X| = m$. *If* $\mathbf{A}$ *is subdirectly irreducible and reverse* k*-definite, but not nilpotent, then* $m \geq 2$, $k \geq 1$ *and* $\mathbf{A}$ *is* (A, X)*-isomorphic to a subautomaton of* $\mathbf{A}(m,k)$.

From this statement the next observation follows immediately.

Corollary 1. *If* $\mathbf{A}$ ($|A| \geq 3$) *is subdirectly irreducible and reverse* k*-definite, but not nilpotent automaton, then there is an* $m \geq 2$ *such that* $\mathbf{A}$ *can be embedded into a quasi-direct product of* $\mathbf{A}(m,k)$ *with a single factor.*

Proposition 3 ([2]). *A generalized definite automaton* $\mathbf{A}$ *with at least three states is subdirectly irreducible if and only if* $Con_e(\mathbf{A}) = \{\rho_1\}$ *or* $Con_e(\mathbf{A}) = \{\sigma_0\}$.

We also need some particular automata. Let $\mathbf{R} = (\{0,1\}, \{x,y\})$ denote the two-state reset automaton defined by $0x^{\mathbf{R}} = 1x^{\mathbf{R}} = 1$ and $0y^{\mathbf{R}} = 1y^{\mathbf{R}} = 0$. Finally, for every positive integer $s \geq 2$, let $\mathbf{I}_s = (\{0, \ldots, s\}, \{x_1, \ldots, x_s\})$ denote

the automaton defined as follows: for all $i \in \{0,1,\ldots,s\}$, $i \neq s-1$ and $x_j \in \{x_1,\ldots,x_s\}$, let

$$ix_j^{\mathbf{I}_s} = \begin{cases} i+j & \text{if } i+j \leq s, \\ s & \text{otherwise,} \end{cases}$$

$$(s-1)x_j^{\mathbf{I}_s} = s-1.$$

It is easy to see that each of the automata defined above is generalized definite.

Now, we are ready to characterize the isomorphically complete systems for $\mathcal{G}$.

Theorem 1. *A system $\mathcal{M}$ of generalized definite automata is isomorphically complete for $\mathcal{G}$ with respect to the α_0-product if and only if there exists an automaton $\mathbf{R}' \in \mathcal{M}$ such that $\mathbf{R}$ can be embedded into a quasi-direct product of $\mathbf{R}'$ with a single factor, moreover, for every positive integer $s \geq 2$, there exists an automaton $\mathbf{I}'_s \in \mathcal{M}$ having at least $s+1$ distinct states denoted by $0,1,\ldots,s$, such that for every $i < j \in \{0,1,\ldots,s\}$, $\{i,j\} \neq \{s-1,s\}$, there exists an input symbol x_{ij} of $\mathbf{I}'_s$ with $ix_{ij} = j$, and there is an input symbol x of $\mathbf{I}'_s$ with $sx = s$ and $(s-1)x = s-1$.*

Proof. To prove the necessity of the conditions, let us suppose that $\mathcal{M}$ is an isomorphically complete system of generalized definite automata for $\mathcal{G}$ with respect to the α_0-product. Since $\mathbf{R} \in \mathcal{G}$, there exist automata $\mathbf{A}_j = (A_j, X_j) \in \mathcal{M}$, $j = 1,\ldots,m$, such that $\mathbf{R}$ can be embedded into an α_0-product $\prod_{j=1}^{m} \mathbf{A}_j(X,\Phi)$. Then it is easy to see that $\mathbf{R}$ can be embedded into a quasi-direct product of some $\mathbf{A}_j$ with a single factor. Similarly, by our assumption, $\mathbf{I}_s$ can be embedded into an α_0-product $\prod_{j=1}^{m} \mathbf{A}_j(X,\Phi)$ of automata in $\mathcal{M}$, since $\mathbf{I}_s \in \mathcal{G}$. Let μ denote a suitable embedding and let $t\mu = (a_{t1},\ldots,a_{tm})$, $t = 0,\ldots,s$. Moreover, let us denote by r the least integer for which $a_{s-1,r} \neq a_{sr}$.

First we show that $a_{ir} \notin \{a_{s-1,r}, a_{sr}\}$, for any $0 \leq i < s-1$. Contrary, let us suppose that $a_{ir} = a_{sr}$ for some $0 \leq i < s-1$. Then there exists an input symbol $y = \varphi_r(a_{i1},\ldots,a_{i,r-1},x_{s-1-i}) \in X_r$ such that $a_{sr}y = a_{ir}y = a_{s-1,r}$. On the other hand, by our assumption, $a_{s-1,r}z = a_{s-1,r}$ and $a_{sr}z = a_{sr}$, where $z = \varphi_r(a_{s1},\ldots,a_{s,r-1},x_1)$. Therefore, $a_{sr}z^h y z^k = a_{s-1,r}$ and $a_{sr}z^h z^k = a_{sr}$ is valid for every pair of integers $h,k \geq 0$. This contradicts the fact that $\mathbf{A}_r$ is a generalized definite automaton. Consequently, $a_{ir} \neq a_{sr}$, for any $0 \leq i < s-1$. One can prove in similar way that $a_{ir} \neq a_{s-1,r}$, for all $i \in \{0,1,\ldots,s-2\}$.

Next we show that the elements a_{tr}, $t = 0,1,\ldots,s-2$, are pairwise different. Contrary, let us suppose that $a_{ir} = a_{jr}$ for some integers $0 \leq i < j \leq s-2$. Since $i < j$ and μ is an embedding, there exists an input symbol $x \in X_r$ such that $a_{ir}x^h = a_{ir}$ holds, for every nonnegative integer h. Moreover, there are such input symbols $y, \bar{x}_1, \bar{x}_2 \in X_r$ for which $a_{ir}y = a_{s-2,r}$, $a_{s-2,r}\bar{x}_1 = a_{s-1,r}$, and $a_{s-2,r}\bar{x}_2 = a_{sr}$. Finally, $a_{sr}z = a_{sr}$ and $a_{s-1,r}z = a_{s-1,r}$, where $z = \varphi_r(a_{s1},\ldots,a_{s,r-1},x_1)$ again. Then $a_{ir}x^h y \bar{x}_1 z^k = a_{s-1,r}$ and $a_{ir}x^h y \bar{x}_2 z^k = a_{s,r}$ hold, for every pair of integers $h,k \geq 0$, which contradicts the fact that $\mathbf{A}_r$ is generalized definite. Therefore, $a_{ir} \neq a_{jr}$, for any integers $0 \leq i < j \leq s-2$. By the two observations given above,

we obtain that the elements $a_{0,r}, a_{1r}, \ldots, a_{sr}$ are pairwise different. This results in that $\mathbf{A}_r$ can be considered as the required automaton $\mathbf{I}'_s$.

In order to prove the sufficiency of the conditions, let us suppose that $\mathcal{M}$ has the required properties. We prove that $\mathcal{M}$ is an isomorphically complete system for $\mathcal{G}$ with respect to the α_0-product. For this reason, let us consider an arbitrary generalized definite automaton $\mathbf{A} = (A, X)$ of n states. We prove by induction on n that $\mathbf{A}$ can be embedded into an α_0-product of automata in $\mathcal{M}$. If $n = 1$ or $n = 2$, then the statement is obviously valid. Now, let $n > 2$ and suppose that the statement is valid for every $m < n$. If $\mathbf{A}$ is subdirectly reducible, then it can be embedded into a direct product of generalized definite automata having fewer states than n. By our induction hypothesis, each component automaton of this direct product can be embedded into an α_0-product of automata in $\mathcal{M}$, and thus, by Lemma 2, $\mathbf{A}$ can be also embedded into an α_0-product of automata in $\mathcal{M}$.

Let us suppose now that $\mathbf{A}$ is subdirectly irreducible. Then, by Proposition 3, $Con_e(\mathbf{A}) = \{\rho_1\}$ or $Con_e(\mathbf{A}) = \{\sigma_0\}$. We distinguish two cases depending on $Con_e(\mathbf{A})$.

Case 1. $Con_e(\mathbf{A}) = \{\rho_1\}$. Let $c \neq d \in A$ with $c\rho_1 d$ for some states $c, d \in A$. Then, by the definition of ρ_1, $cx = dx$, for all $x \in X$. Let $X_1 = \{x : x \in X \text{ and } cx = c\}$. Moreover, let 0, 1 and u, v denote the states and the input symbols of $\mathbf{R}'$ for which $0u = 1u = 0$ and $0v = 1v = 1$ hold. Let us consider the α_0-product $\mathbf{A}/\rho_1 \times \mathbf{R}'(X, \Phi)$ defined as follows. For all $a \in A \setminus \{c, d\}$ and $x \in X$, let

$$\varphi_1(x) = x$$

and

$$\varphi_2(\{a\}, x) = \begin{cases} u & \text{if } ax^{\mathbf{A}} \notin \{c, d\} \text{ or } ax^{\mathbf{A}} = c, \\ v & \text{otherwise,} \end{cases}$$

$$\varphi_2(\rho_1(c), x) = \begin{cases} u & \text{if } cx^{\mathbf{A}} \notin \{c, d\} \text{ or } x \in X_1, \\ v & \text{otherwise.} \end{cases}$$

Let us define the mapping $\mu : A \to A/\rho_1 \times \{0, 1\}$ by

$\mu : a \to (\{a\}, 0)$, for all $a \in A \setminus \{c, d\}$,
$\mu : c \to (\rho_1(c), 0)$,
$\mu : d \to (\rho_1(c), 1)$,

where $\rho_1(c)$ denotes the equivalence class containing c. Then it is easy to see that μ is an embedding of $\mathbf{A}$ into the α_0-product $\mathbf{A}/\rho_1 \times \mathbf{R}'(X, \Phi)$, and thus, our induction hypothesis and Lemmas 1 and 2 imply that $\mathbf{A}$ can be embedded into an α_0-product of automata in $\mathcal{M}$.

Case 2. $Con_e(\mathbf{A}) = \{\sigma_0\}$. Let $c \neq d \in A$ with $c\sigma_0 d$. Then $cx = c$ and $dx = d$, for all $x \in X$. Let us suppose that $\mathbf{A}$ is weakly (h, k)-definite for some $h \geq 0$,

$k \geq 0$. For all $a \in A$ and $u \in X^h$, let us define the subautomaton $\mathbf{A}_{au} = (A_{au}, X)$, where $A_{au} = \{aup : p \in X^*\}$. Then $\mathbf{A}_{au}$ is a weakly k-definite automaton. Indeed, let $v \in X^k$ be an arbitrary word and $a' \in A_{au}$. Then there is a word $p \in X^*$ such that $a' = aup$. Since $\mathbf{A}$ is weakly (h,k)-definite, $a'v = aupv = auv$ is valid for all $a' \in A_{au}$, and hence, $A_{au}v = \{auv\}$.

Now, we distinguish two subcases depending on the subautomata $\mathbf{A}_{au}$.

Subcase 1. There exist a state $a \in A$ and a word $u \in X^h$ such that $\mathbf{A}_{au}$ is not singleton. Then $\mathbf{A}_{au}$ is a subdirectly irreducible definite subautomaton of $\mathbf{A}$, since for any congruence γ of $\mathbf{A}_{au}$, the relation $\gamma \cup \omega_A$ is a congruence relation of $\mathbf{A}$. If $|A_{au}| \geq 3$, then by Proposition 1, $Con_e(\mathbf{A}_{au}) = \{\rho_1'\}$, where ρ_1' denotes the corresponding relation belonging to $\mathbf{A}_{au}$. Then there are states $e, f \in A_{au}$ such that $ex = fx$, for all $x \in X$. Then it is easy to see that the relation θ defined on A by

$$a' \Theta a'' \text{ if and only if } \{a', a''\} \subseteq \{e, f\} \text{ or } a' = a''$$

is an elementary congruence of $\mathbf{A}$ distinct from σ_0 which contradicts our assumption that $Con_e(\mathbf{A}) = \{\sigma_0\}$. If $|A_{au}| = 2$, then $A_{au} = \{e, f\}$ for some states $e, f \in A$ and $ex = fx$ is valid, for all $x \in X$. Then one can define Θ in the same way as above, and Θ is an elementary congruence of $\mathbf{A}$ which results in a contradiction again. Consequently, this subcase is impossible.

Subcase 2. For all $a \in A$ and $u \in X^h$, $\mathbf{A}_{au}$ is singleton. Then, $aux = au$, for all $u \in X^h$ and $x \in X$, and hence, $\mathbf{A}$ is weakly reverse h-definite, moreover, by $Con_e(\mathbf{A}) = \{\sigma_0\}$, $\mathbf{A}$ is not nilpotent. Since $\mathbf{A}$ is subdirectly irreducible, by Corollary 1, $\mathbf{A}$ can be embedded into a quasi-direct product of $\mathbf{A}(m,h)$ with a single factor for an integer $m \geq 2$. Without loss of generality, we may assume that h is the least integer with this property. Let τ denote a suitable embedding of $\mathbf{A}$ into the quasi-direct product of $\mathbf{A}(m,h)$ with a single factor. Let $B_i = (A\tau \cap A(m,i))\tau^{-1}$, $i = 0, 1, \ldots, h$. Since τ is an embedding, it is easy to show that

(3) $\{c,d\} = B_0 \subset B_1 \subset \ldots \subset B_h = A$

(4) $B_i x^{\mathbf{A}} \subseteq B_{i-1}$, for all $1 \leq i \leq h$ and $x \in X$.

Let us consider now the α_0-product $\mathbf{A}/\sigma_0 \times \mathbf{I}'_{h+1}(X, \Phi)$ defined as follows. For all $a \in A \setminus \{c,d\}$ and $x \in X$, let

$\varphi_1(x) = x$,

$$\varphi_2(a,x) = \begin{cases} x_{h-j,h-i} & \text{if } a \in B_j \setminus B_{j-1} \text{ and } ax^{\mathbf{A}} \in B_i \setminus B_{i-1} \\ & \text{for some } 1 \leq i < j \leq h, \\ x_{h-j,h} & \text{if } a \in B_j \setminus B_{j-1} \text{ and } ax^{\mathbf{A}} = c \text{ for some } 1 \leq j \leq h, \\ x_{h-j,h+1} & \text{if } a \in B_i \setminus B_{i-1} \text{ and } ax^{\mathbf{A}} = d \text{ for some } 1 \leq j \leq h, \end{cases}$$

$\varphi_2(\{c,d\}, x) = x'$,

where x' denotes now the input symbol of $\mathbf{I}_{h+1}$ for which $hx' = h$ and $(h+1)x' = h+1$ hold. By (3) and (4), the feedback functions are well-defined.

Let us define the mapping μ of $A \to A/\sigma_0 \times \{0, 1, \ldots h+1\}$ as follows. For every $a \in A \setminus \{c, d\}$, let

$\mu : a \to (\{a\}, h - j)$ if $a \in B_j \setminus B_{j-1}$ for some $1 \leq j \leq h$,
$\mu : c \to (\{c, d\}, h)$,
$\mu : d \to (\{c, d\}, h + 1)$.

Now, it is easy to check that μ is an embedding of $\mathbf{A}$ into the α_0-product under consideration. Then, the induction assumption and Lemmas 1 and 2 yield that $\mathbf{A}$ can be embedded into an α_0-product of automata in $\mathcal{M}$. This ends the proof of the statement.

From Theorem 1, the next observation follows.

Corollary 2. *There is no finite system $\mathcal{M}$ of generalized definite automata which is isomorphically complete for $\mathcal{G}$ with respect to the α_0-product.*

The following statement shows that we can obtain finite isomorphically complete systems by allowing automata as components which are not necessarily generalized definite.

Theorem 2. *A system $\mathcal{M}$ of automata is isomorphically complete for $\mathcal{G}$ with respect to the α_0-product if and only if $\mathcal{M}$ satisfies the conditions below:*

(1) *there exists an automaton $\mathbf{R}' \in \mathcal{M}$ such that $\mathbf{R}$ can be embedded into a quasi-direct product of $\mathbf{R}'$ with a single factor,*

(2) (a) *for every positive integer $s \geq 2$, there exists an automaton $\mathbf{I}'_s \in \mathcal{M}$ which has $s+1$ distinct states, denoted by $0, 1, \ldots, s$, such that for all $i < j \in \{0, 1, \ldots, s\}$, $\{i, j\} \neq \{s-1, s\}$, there exists an input symbol x_{ij} of $\mathbf{I}'_s$ with $ix_{ij} = j$, and there is a further input symbol x of $\mathbf{I}'_s$ such that $sx = s$ and $s - 1x = s - 1$,*

or

(b) *there exists an $\mathbf{A} \in \mathcal{M}$ which has a state a and not necessarily distinct input symbols u, v, w, z such that $au = a$, $av \neq aw$, $avz = av$, and $awz = aw$.*

Proof. To prove the necessity of the conditions, let us suppose that $\mathcal{M}$ is isomorphically complete for $\mathcal{G}$ with respect to the α_0-product. The necessity of (1) follows from the proof of Theorem 1.

Now, we prove that (2)(a) or (2)(b) is valid for $\mathcal{M}$. For this purpose, let us suppose that (2)(a) is not valid. Then there is a positive integer $s_0 \geq 2$ such that $\mathbf{I}'_{s_0} \notin \mathcal{M}$. By our assumption, $\mathbf{I}_{s_0}$ can be embedded into an α_0-product $\prod_{j=1}^{m} \mathbf{A}_j(X, \Phi)$ of automata in $\mathcal{M}$ since $\mathbf{I}_{s_0} \in \mathcal{G}$. Let μ denote a suitable isomorphism and let $t\mu = (a_{t1}, \ldots, a_{tm})$, $t = 1, \ldots, s_0$. Let r be the least integer for which

$a_{s_0-1,r} \neq a_{s_0,r}$. Let us observe that if the elements $a_{0r}, a_{1r}, \ldots, a_{s_0,r}$ are pairwise different, then $\mathbf{A}_r$ can be considered as $\mathbf{I}'_{s_0}$ which is a contradiction. Consequently, there are $i < j \in \{0, 1, \ldots, s_0\}$, $\{i, j\} \neq \{s_0 - 1, s_0\}$ such that $a_{ir} = a_{jr}$. Now, it is easy to show that $\mathbf{A}_r$ satisfies condition (b), and therefore, (2)(b) is valid for $\mathcal{M}$.

In order to prove the sufficiency, let us suppose that $\mathcal{M}$ satisfies the conditions. If (1) and (2)(a) are valid, then Theorem 1 implies that $\mathcal{M}$ is isomorphically complete for $\mathcal{G}$ with respect to the α_0-product. Let us assume now that (1) and (2)(b) are valid for $\mathcal{M}$ with $\mathbf{R}'$ and $\mathbf{A}$, respectively. Then, by Lemma 2 and the sufficiency of conditions (1) and (2)(a), it is sufficient to show that $\mathbf{I}_s$ can be embedded into an α_0-product of automata in $\{\mathbf{R}, \mathbf{A}\}$, for every positive integer $s \geq 2$. For this purpose, let $s \geq 2$ be an arbitrary positive integer. Let us define the α_0-product

$$\mathbf{R} \times \cdots \times \mathbf{R} \times \mathbf{A}(\{x_1, \ldots, x_s\}, \Phi),$$

where the number of the occurences of $\mathbf{R}$ is equal to $s - 1$, in the following way. For every $(r_1, \ldots, r_{s-1}) \in \{0,1\}^{s-1}$, $x_k \in \{x_1, \ldots, x_s\}$, and $j \in \{2, \ldots, s-1\}$, let

$$\varphi_1(x_k) = x,$$

$$\varphi_j(r_1, \ldots, r_{j-1}, x_k) = \begin{cases} x & \text{if } \sum_{t=1}^{j-1} r_t + k \geq j, \\ y & \text{otherwise,} \end{cases}$$

$$\varphi_s(r_1, \ldots, r_{s-1}, x_k) = \begin{cases} u & \text{if } \sum_{t=1}^{s-1} r_t + k < s-1, \\ v & \text{if } \sum_{t=1}^{s-1} r_t + k = s-1, \\ w & \text{if } \sum_{t=1}^{s-1} r_t + k \geq s \text{ and } \sum_{t=1}^{s-1} r_t \neq s-1, \\ z & \text{otherwise,} \end{cases}$$

where a and u, v, w, z denote the suitable state and input symbols of $\mathbf{A}$, respectively. Now, let us consider the mapping μ defined by

$$\begin{aligned} &\mu : 0 \to (0, 0, 0, \ldots, 0, a), \\ &\mu : 1 \to (1, 0, 0, \ldots, 0, a), \\ &\mu : 2 \to (1, 1, 0, \ldots, 0, a), \\ &\quad \vdots \\ &\mu : s-2 \to (1, 1, \ldots 1, 0, a), \\ &\mu : s-1 \to (1, 1, \ldots, 1, 1, av), \\ &\mu : s \to (1, 1, \ldots, 1, 1, aw). \end{aligned}$$

Then it is easy to see that μ is an isomorphism of $\mathbf{I}_s$ into the α_0-product $\mathbf{R} \times \cdots \times \mathbf{R} \times \mathbf{A}(\{x_1, \ldots, x_s\}, \Phi)$ under consideration which ends the proof of Theorem 2.

The next statement directly follows from the definition of the α_i-products.

Lemma 3. *If the automaton* $\mathbf{A}$ *can be embedded into an* α_0*-product of automata* $\mathbf{A}_j$, $j = 1, \ldots, n$, *and each* $\mathbf{A}_j$ *can be embedded into an* α_1*-product of automata* $\mathbf{A}_{jt}$, $t = 1, \ldots, m_j$, *then* $\mathbf{A}$ *can be embedded into an* α_1*-product of automata* $\mathbf{A}_{jt}$, $t = 1, \ldots, m_j$; $j = 1, \ldots, n$.

Now, let $i \geq 1$ be an arbitrary integer. Then the isomorphically complete systems of generalized definite automata with respect to the α_i-product can be characterized as follows.

Theorem 3. *A system $\mathcal{M}$ of generalized definite automata is isomorphically complete for $\mathcal{G}$ with respect to the α_i-product ($i \geq 1$) if and only if there exists an automaton $\mathbf{R}'' \in \mathcal{M}$ such that $\mathbf{R}''$ has two distinct states denoted by $0, 1$ and four not necessarily distinct input symbols v, x, y, z such that $1v = 0$, $0x = 0$, $0y = 1$, $1z = 1$.*

Proof. The necessity of the conditions can be proved in a similar way as in the case of Theorem 1.

Regarding the sufficience, let us observe, that $\mathbf{R}$ can be embedded into an α_i-product of $\mathbf{R}''$ with a single factor, and this product is an α_1-product of $\mathbf{R}''$ with single factor. Now we show that for every integer $s \geq 2$, the automaton $\mathbf{I}_s$ can be embedded into an α_1-power of $\mathbf{R}''$.

Let $s \geq 2$ be an arbitrary integer. Let us consider the α_1-power $(\mathbf{R}'')^s(\{x_1, \ldots, x_s\}, \Phi)$ defined as follows. For all $1 \leq k < s$, $(v_1, \ldots, v_k) \in \{0,1\}^k$, and $x_j \in \{x_1, \ldots, x_s\}$, let

$$\varphi_k(v_1, \ldots, v_k, x_j) = \begin{cases} x & \text{if } j + \sum_{t=1}^{k} v_t < k, \\ y & \text{if } j + \sum_{t=1}^{k} v_t \geq k \text{ and } v_k = 0, \\ z & \text{otherwise,} \end{cases}$$

furthermore, for all $(v_1, \ldots, v_s) \in \{0,1\}^s$ and $x_j \in \{x_1, \ldots, x_s\}$, let

$$\varphi_s((v_1, \ldots, v_s, x_j) = \begin{cases} x & \text{if } \sum_{t=1}^{s} v_t = s-1, \\ x & \text{if } \sum_{t=1}^{s} v_t < s-1 \text{ and } j + \sum_{t=1}^{s} v_t \leq s-1, \\ y & \text{if } \sum_{t=1}^{s} v_t < s-1 \text{ and } j + \sum_{t=1}^{s} v_t \geq s, \\ z & \text{if } \sum_{t=1}^{s} v_t = s. \end{cases}$$

Then it is easy to see, that the mapping μ given by

$$\begin{aligned} &\mu : 0 \to (0,0,0,\ldots,0), \\ &\mu : 1 \to (1,0,0,\ldots,0), \\ &\mu : 2 \to (1,1,0,\ldots,0), \\ &\quad\vdots \qquad\qquad\quad \vdots \\ &\mu : (s-1) \to (1,1,\ldots,1,0), \\ &\mu : s \to (1,1,\ldots,1,1), \end{aligned}$$

is an embedding of $\mathbf{I}_s$ into the α_1-power under consideration.

By Theorem 1, the system $\mathcal{K} = \{\mathbf{R}\} \cup \{\mathbf{I}_s : s = 2, 3, \ldots\}$ is an isomorphically complete system for $\mathcal{G}$ with respect to the α_0-poduct. Therefore, every generalized

definite automaton can be embedded into an α_0-product of automata in $\mathcal{K}$. On the other hand, we have proved that every automaton in $\mathcal{K}$ can be embedded into an α_1-power of $\mathbf{R}''$. Then, by Lemma 3, we obtain that every generalized definite automaton can be embedded into an α_1-power of $\mathbf{R}''$, and consequently, $\{\mathbf{R}''\}$, and also $\mathcal{M}$, are isomorphically complete systems for $\mathcal{G}$ with respect to the α_i-product.

The following assertion shows that we obtain the same characterization of the isomorphically complete systems consisting of not necessarily generalized definite automata with respect to the α_i-product ($i \geq 1$).

Theorem 4. *A system $\mathcal{M}$ of automata is isomorphically complete for $\mathcal{G}$ with respect to the α_i-product ($i \geq 1$) if and only if it contains an automaton $\mathbf{R}''$ such that $\mathbf{R}''$ has two distinct states, denoted by 0, 1, and four not necessarily distinct input symbols x, y, z, v with $0x^{\mathbf{R}''} = 0$, $0y^{\mathbf{R}''} = 1$, $1z^{\mathbf{R}''} = 1$, $1v^{\mathbf{R}''} = 0$.*

Proof. The validity of Theorem 4 follows immediately from Theorem 3.

References

[1] Burris, S., H. P. Sankappanavar, *A Course in Universal Algebra*, Springer-Verlag, New York-Berlin, 1981.

[2] Ćirić, M., B. Imreh, M. Steinby, Subdirectly irredicible definite, reverse definite, and generalized definite automata, *Publ. Electrotechn. Fak. Ser. Mat.*, **10** (1999), 69-79.

[3] Gécseg, F., Composition of automata, Proceedings of the 2nd Colloquium on Automata, Languages and Programming, Saarbrücken, LNCS **14** (1974), 351-363.

[4] Gécseg, F., *Products of Automata*, Springer-Verlag, Berlin - Heidelberg- New York - Tokyo (1986).

[5] Ginzburg, A., About some properties of definite, reverse-definite and related automata, *IEEE Trans. Electronic Computers* **EC-15** (1966), 809-810.

[6] Grätzer, G., *Universal Algebra*, 2nd edn. Springer-Verlag, New York-Berlin-Heidelberg-Tokyo, 1979.

[7] Petković, T., M. Ćirić, S. Bogdanović, Decomposition of automata and transition semigroups, *Acta Cybernetica* **13** (1998), 385-403.

[8] Steinby, M., On definite automata and related systems, *Ann. Acad. Sci. Fenn.*, Ser. A I, **444** (1969).

Received October, 2000

Acta Cybernetica **15** (2001) 45-52.

Languages Recognized by a Class of Finite Automata *

A.V. Kelarev † O.V. Sokratova ‡

Abstract

We consider automata defined by left multiplications in graph algebras, and describe all languages recognized by these automata in terms of combinatorial properties of words which belong to these languages, regular expressions and linear grammars defining these languages. This description is applied to investigate closure properties of the obtained family of languages.

A *language* over an alphabet X is a subset of the free monoid X^* generated by X. We use standard concepts of automata and languages theory (see [7], [11]).

Graph algebras make it possible to apply methods of universal algebra to various problems of discrete mathematics and computer science. They have been investigated by several authors (see [2], [4], [5], [6], [8], [9], and [10] for references). Throughout the word *graph* means a finite directed graph without multiple edges but possibly with loops. The *graph algebra* $\mathrm{Alg}(D)$ of a graph $D = (V, E)$ is the set $V \cup \{\infty\}$ equipped with multiplication given by the rule

$$xy = \begin{cases} x & \text{if } (x, y) \in E, \\ \infty & \text{otherwise,} \end{cases}$$

for all $x, y \in V$. In this paper we use graph algebras as language recognizers.

Let $\mathrm{Alg}(D)$ be a graph algebra. Put $\mathrm{Alg}^1(D) = \mathrm{Alg}(D) \cup \{1\}$, and extend the multiplication of $\mathrm{Alg}(D)$ to the whole set $\mathrm{Alg}^1(D)$ by assuming that 1 acts as an identity on all elements of $\mathrm{Alg}^1(D)$. Let T be a subset of $\mathrm{Alg}^1(D)$, and let $f\colon X \to \mathrm{Alg}^1(D)$ be any mapping. We consider the *graph algebra automaton* $\mathrm{Atm}(D, T)$, where

- the set of states is $\mathrm{Alg}^1(D)$;
- 1 is the initial state;

*The author has been supported by the Estonian Science Foundation, Grant 4912.

†Department of Mathematics, University of Tasmania, P.O. Box 252-37, Hobart, Tasmania 7001, Australia, e-mail: Andrei.Kelarev@utas.edu.au

‡Institute of Computer Science, University of Tartu, Liivi 2, 50409 Tartu, Estonia, e-mail: olga@cs.ut.ee

- T is the set of terminal states;
- the next-state function is given by $a \cdot x = f(x)a$, for $a \in \mathrm{Alg}^1(D)$, $x \in X$.

This automaton is defined by left multiplications of elements of the graph algebra.

Our main theorem describes all languages recognized by graph algebra automata in terms of combinatorial properties of words which belong to these languages, as well as in terms of regular expressions for these languages or their complements (see Theorem 1). This description allows us to answer several natural questions concerning the class $\mathcal{G}$ of languages recognized by graph algebra automata. First, we show that this class is not empty and, moreover, contains certain fairly large subclasses (see Corollary 4). Second, we verify that $\mathcal{G}$ is a proper subclass of the class of languages recognizable by finite state automata (see Corollary 6). Third, we give examples which demonstrate that the whole $\mathcal{G}$ is not closed for union, intersection, and product (see Example 7). However, we represent $\mathcal{G}$ as a union of two classes $\mathcal{G}_a$ and $\mathcal{G}_b$ such that $\mathcal{G}_a$ is closed under intersection and left derivative, and $\mathcal{G}_b$ is closed under union and right derivative. Finally, we show that the whole class $\mathcal{G}$ is closed under the Kleene $*$-operation and complement (see Corollaries 8 and 9).

Theorem 1 *For any language L over an alphabet X, the following are equivalent:*

(i) *L is recognized by a graph algebra automaton;*

(ii) *at least one of the following two conditions is satisfied for all $x, y \in X$, and $u, v \in X^*$:*

(a) *$xyu \in L$ implies $yu \in L$, and*

$xu, yxv \in L$ implies $yxu \in L$;

(b) *$yu \in L$ implies $xyu \in L$, and*

$yxu \in L$ implies $xu \in L$ or $yxv \in L$.

(iii) *there exist disjoint subsets X_1, X_2 of X and a relation $G \subseteq X \times X$ such that the language $L \setminus \{1\}$ or $X^+ \setminus L$ has the following regular expression:*

$$X^*X_1X^* + X^*X_2 + \sum_{(x_j,x_i)\in G} X^*x_ix_jX^*; \tag{1}$$

(iv) *there exist subsets $Q \subseteq X$ and $P \subseteq X \times X$ such that the language $L \setminus \{1\}$ or $X^+ \setminus L$ is generated by the right linear grammar with the alphabet X, the set $W = \{x' \mid x \in X\} \cup \{s_0\}$ of nonterminal symbols, the start symbol s_0, and productions*

$$\begin{aligned} s_0 &\to xx' \quad \text{for all } x \in X, \\ x' &\to yy' \quad \text{for all } (y,x) \in P, \\ x' &\to 1 \quad \text{for all } x \in Q. \end{aligned} \tag{2}$$

Proof.

(ii)⇒(i): First, suppose that the language L satisfies (a). Introduce a graph $D = (V, E)$ with the set $V = X$ of vertices, and the set E of edges consisting of all pairs (x, y) such that $yxu \in L$ for some $u \in X^*$. Let f be the mapping from X to $\mathrm{Alg}^1(D)$ defined by $f(x) = x$. Put $T = (\{1\} \cup X) \cap L$.

Take an arbitrary word $u = x_1 \cdots x_n$ in L. If $u = 1$, i.e. $n = 0$, then $1 \in T$, and the automaton $\mathrm{Atm}(D, T)$ accepts u. Further, assume that $n > 0$. The first implication of condition (a) shows that $x_k \cdots x_n \in L$, for all $k = 1, \ldots, n$. It follows that $(x_2, x_1), (x_3, x_2), \ldots, (x_n, x_{n-1}) \in E$ by the definition of E, and $x_n \in T$ by the definition of T. We get $1 \cdot u = f(x_n)(\cdots f(x_1)) = f(x_n) = x_n \in T$, and so the automaton $\mathrm{Atm}(D, T)$ recognizes u.

Consider any word $u = x_1 \cdots x_n$ accepted by $\mathrm{Atm}(D, T)$. If $u = 1$, then $1 \in T$, and so $u = 1 \in L$. Further, assume that $n > 0$. As above, $1 \cdot u \in T$ means that $x_n, \ldots, x_1$ is a directed path in D and $x_n \in T$. By reversed induction on k we show that $x_k \cdots x_n \in L$, for all $k = 1, \ldots, n$. If $k = n$, then $x_n \in T \subseteq L$ by definition. Assume that $x_k x_{k+1} \cdots x_n \in L$, for some $1 < k \leq n$. Since $(x_k, x_{k-1}) \in E$, we have $x_{k-1} x_k v \in L$, for some $v \in X^*$. Then the second implication of (a) yields $x_{k-1} x_k \cdots x_n \in L$, as required. Therefore $u = x_1 \cdots x_n \in L$. Thus L is the language recognized by $\mathrm{Atm}(D, T)$.

Second, suppose that L satisfies condition (b). Observe that the complement $\overline{L} = X^* \setminus L$ of L satisfies condition (a) if and only if L satisfies condition (b). Indeed, denoting the logical negation of a proposition P by $\overline{P}$, we get

$$
\begin{aligned}
&(xyu \notin L \Rightarrow yu \notin L) \equiv \\
&\quad \equiv \overline{xyu \in L} \Rightarrow \overline{yu \in L}) \\
&\quad \equiv (\overline{\overline{xyu \in L}} \vee \overline{yu \in L}) \\
&\quad \equiv (xyu \in L \vee \overline{yu \in L}) \\
&\quad \equiv (yu \in L \Rightarrow xyu \in L) \quad \text{and} \\
&(xu, yxv \notin L \Rightarrow yxu \notin L) \equiv \\
&\quad \equiv (\overline{xu \in L} \wedge \overline{yxv \in L} \Rightarrow \overline{yxu \in L}) \\
&\quad \equiv (\overline{\overline{xu \in L} \wedge \overline{yxv \in L}} \vee \overline{yxu \in L} \\
&\quad \equiv (xu \in L \vee yxu \in L \vee \overline{yxv \in L}) \\
&\quad \equiv (yxu \in L \Rightarrow xu \in L \vee yxv \in L).
\end{aligned}
$$

Therefore $\overline{L} = X^* \setminus L$ satisfies (a), and so it is recognized by some automaton $\mathrm{Atm}(D, T)$. Hence L is recognized by the automaton $\mathrm{Atm}(D, \overline{T})$, where $\overline{T} = \mathrm{Alg}^1(D) \setminus T$.

(i)⇒(iii): Suppose that L is recognized by a graph algebra automaton $\mathrm{Atm}(D, T)$ of a graph $D = (V, E)$. First, assume that $\infty \in T$. Let us define the sets:

$$X_1 = \{x \in X \mid f(x) = \infty\} \qquad X_2 = \{x \in X \mid f(x) \in T \setminus \{\infty\}\}$$

and the relation

$$G = \{(x_i, x_j) \in X \times X \mid (f(x_i), f(x_j)) \notin E\}.$$

Take an element $u = x_1 \cdots x_n$ in $L \setminus \{1\}$. Since $u \neq 1$, we get $n > 0$. By the definition, $1 \cdot u = f(x_n)(\cdots f(x_1)) \in T$. The following three cases are possible:

Case 1: $1 \cdot u \in T \setminus \{\infty\}$. Then $f(x_n) = 1 \cdot u \in T \setminus \{\infty\}$, and so $u \in X^*X_2$.

Case 2: $1 \cdot u = \infty$ and $f(x_i) = \infty$, for some $i = 1, \ldots, n$. Then $u \in X^*X_1X^*$.

Case 3: $1 \cdot u = \infty$ and all $f(x_i) \neq \infty$. Then there exists $i = 1, \ldots, n-1$ such that $(f(x_{i+1}), f(x_i)) \notin E$. It follows that $u \in X^*x_ix_{i+1}X^*$.

On the other hand, consider an arbitrary element $u = x_1 \cdots x_n$ of the language defined by (1). If $u \in X^*X_1X^*$ or $u \in X^*x_ix_{i+1}X^*$, then $1 \cdot u = \infty$, and so $u \in L$. If $u \in X^*X_2$, then either $1 \cdot u = \infty \in T$, and so $u \in L$, or $1 \cdot u = f(x_n) \in T \setminus \{\infty\}$, and $u \in L$, again.

Thus $L \setminus \{1\}$ is given by the regular expression (1).

Second, assume that $\infty \notin T$. Then the complement of L is defined by the regular expression (1).

(iii)⇒(ii): Let L be a language defined by the regular expression (1). We are going to verify condition (b) of Theorem 1. If $yu \in L$, then obviously $xyu \in L$. Now, suppose that $yxu \in L$. First, assume that $yxu \in X^*X_1X^*$. If $y \in X_1$, then $yxv \in X_1X^* \subseteq L$. Otherwise, $xu \in X^*X_1X^* \subseteq L$. Second, assume that $yxu \in X^*X_2$. Then $xu \in X^*X_2 \subseteq L$. Third, assume that $yxu \in X^*x_ix_jX^*$, for some $(x_j, x_i) \in G$. If $y = x_i$, then $yxv \in x_ix_jX^* \subseteq L$. Otherwise, $xu \in X^*x_ix_jX^* \subseteq L$. Thus condition (b) of Theorem 1 holds, and hence L is recognized by a graph algebra automaton.

(i)⇒(iv): Suppose that the graph algebra automaton $\mathrm{Atm}(D, T)$ of a graph $D = (V, E)$ recognizes L. First, assume that $\infty \notin T$. The standard method gives us a right linear grammar which generates L (see, for example, the proof of Proposition 6.2.3 in [3]). Removing redundant productions from this grammar, and simplifying notation of states, we get the right linear grammar mentioned in condition (iv) with $Q = T$ and $P = E$. Note that s_0 is a nontermainal state. Thus this grammar generates the same language.

Second, assume that $\infty \in T$. It can be easily seen that the grammar specified in (iv) generates $X^+ \setminus L$.

(iv)⇒(i): Suppose that $L \setminus \{1\}$ is generated by the right linear grammar given in (iv), and so we are given Q, P and T. Consider the graph $D = (V, E)$ with the set $V = X$ of vertices and the set $E = P$ of edges. Let f be the mapping from X to $\mathrm{Alg}^1(D)$ defined by $f(x) = x$. Put $T = Q$. It is routine to verify that L is recognized by the graph algebra automaton $\mathrm{Atm}(D, T)$ of D.

Suppose that $\overline{L} \setminus \{1\} = X^* \setminus (L \cup \{1\})$ is generated by the right linear grammar specified in (iv). Then $\overline{L}$ is recognized by the automaton $\mathrm{Atm}(D, T)$, and hence L is recognized by the automaton $\mathrm{Atm}(D, \overline{T})$, where $\overline{T} = \mathrm{Alg}^1(D) \setminus T$. □

Corollary 2 *It is decidable whether a regular language belongs to the class $\mathcal{G}$.*

Proof. follows from condition (iii) of our main theorem. Indeed, given a regular language, for all disjoint subsets X_1, X_2 of X and all relations $G \subseteq X \times X$, we can use well known algorithms to verify whether the regular language (1) is equal to $L \setminus \{1\}$ or $X^+ \setminus L$. □

Denote by $\mathcal{G}_a$ the subclass of $\mathcal{G}$ containing the languages over X satisfying condition (a) of Theorem 1 and by $\mathcal{G}_b$ the subclass of $\mathcal{G}$ containing the languages over X satisfying condition (b). Let $\tilde{\mathcal{G}}_i = \{L \subseteq X^* \mid \overline{L} = X^* \setminus L \in \mathcal{G}_i\}$, for $i = a, b$. It follows from the proof of Theorem 1 that

$$\tilde{\mathcal{G}}_a = \mathcal{G}_b \qquad \text{and} \qquad \tilde{\mathcal{G}}_b = \mathcal{G}_a, \tag{3}$$

the regular expression (1) describes languages of the class $\mathcal{G}_b$, and the right linear grammar specified in condition (iv) of Theorem 1 describes languages of the class $\mathcal{G}_a$.

Corollary 3 *A language L belongs to the class $\mathcal{G}_a \cap \mathcal{G}_b$ if and only if there exists a subset X_2 of X such that $L \setminus \{1\}$ is given by the regular expression:*

$$X^* X_2 \tag{4}$$

Proof. It is easily seen that the regular expression (4) is a particular case of the regular expression (1), and so every language defined by it belongs to $\mathcal{G}_b$.

Further, it is easy to verify that every language defined by the regular expression (4) satisfies condition (a) of Theorem 1, and therefore belongs to $\mathcal{G}_a$. Thus every language described by the regular expression (4) belongs to the class $\mathcal{G}_a \cap \mathcal{G}_b$

Conversely, consider an arbitrary language L of the class $\mathcal{G}_a \cap \mathcal{G}_b$. By Theorem 1, there exist disjoint subsets X_1, X_2 of X and a relation $G \subseteq X \times X$ such that $L \setminus \{1\}$ has the regular expression (1). Moreover, we can choose $X_1 = \{x \in X \mid f(x) = \infty\}$, $X_2 = \{x \in X \mid f(x) \in T \setminus \{\infty\}\}$ and $G = \{(x_i, x_j) \in X \times X \mid (f(x_i), f(x_j)) \notin E\}$, where L is recognized by a graph algebra automaton $\mathrm{Atm}(D, T)$ of a graph $D = (V, E)$ such that $\infty \in T$.

Since the language X^+ obviously has a regular expression of the form (4), we may assume that L is not equal to any of the languages X^+ and X^*.

First, suppose that $X_1 \neq \emptyset$. Take any $x \in X_1$ and $u \in X^+$. Condition (a) of Theorem 1 implies $u \in L$. This contradiction shows that $X_1 = \emptyset$.

Next, suppose that $G \neq \emptyset$. Choose a pair $(x_j, x_i) \in G$ and $u \in X^+$. Then $x_i x_j u \in L$, and condition (a) of Theorem 1 implies $u \in L$, a contradiction. Hence $G = \emptyset$.

Therefore $L \setminus \{1\}$ is given by the regular expression (4). □

In particular, the class $\mathcal{G}_a \cap \mathcal{G}_b$ contains the languages $\emptyset$, $\{1\}$, X^+, and X^*.

Corollary 4 *The class $\mathcal{G}_a$ contains all regular languages given by the regular expressions of the form*

$$X_1 X_2^* X_3 X_4^* \cdots X_{2n}^* X_{2n+1}, \tag{5}$$

where $X_1, X_2, \ldots, X_{2n+1} \subseteq X \cup \{1\}$, *and* $X_i \cap X_j = \{1\}$, *for all* $1 \leq i < j \leq 2n+1$.

Proof. Let L be a language defined by the regular expression (5). First, suppose that $xyu \in L$. Since the empty word 1 belongs to all sets X_i, it follows that $yu \in L$. Second, suppose that $xu, yxv \in L$. Since 1 is the only common element of X_i and X_j for $i \neq j$, we see that both occurrences of the letter x in xu and in yxv come from the same set X_i, where $1 \leq i \leq 2n+1$. It follows that $yxu \in L$. Thus condition (a) of Theorem 1 holds, and therefore $L \in \mathcal{G}_a$. □

Next, we give an example of a language, which belongs to $\mathcal{G}_a$ but cannot be defined by a regular expression of the form (5).

Example 5 Let $X = \{x_1, x_2, \ldots, x_n\}$, and let L be the set of all words x_i^k such that k is a positive integer, and $1 \leq i \leq n$. It is easily seen that L satisfies condition (a) of Theorem 1, and so $L \in \mathcal{G}_a$. However, for $n > 1$, it is clear that L cannot be described by an expression of the form (5), because all languages described by these expressions have a word containing all the letters $x_1, \ldots, x_n$.

It is easily seen that all languages in the class $\mathcal{G}_b$, except $\emptyset$ and $\{1\}$, are infinite. On the other hand, the class $\mathcal{G}_a$ contains some finite languages, but not all, as the following corollary shows.

Corollary 6 *Let* L *be a finite language over an alphabet* X, *where* $|X| = n$. *If* $L \in \mathcal{G}_a$, *then* $|L| \leq 2^n$.

Proof. First, we show that L has no words with two occurrences of the same letter. Suppose to the contrary that L contains a word $w = axbxc$, where $x \in X$, $a, b, c \in X^*$. Since L satisfies condition (a) of Theorem 1, it follows that L contains all words $axbxbxc, axbxbxbxc, \ldots$. This contradicts the finiteness of L.

Second, we show that if two letters x_i, x_j occur together in several words of L, then they occur in the same order in all of these words. Suppose to the contrary that L contains words $w = ax_1bx_2c$ and $w' = a'x_2b'x_1c'$, where $x_1, x_2 \in X$, $a, b, c, a', b', c' \in X^*$. It follows from the second implication of condition (a) of Theorem 1, that L contains the word $w = ax_1bx_2b'x_1c'$, a contradiction.

Therefore every word of L is defined by the set of its letters. Thus

$$|L| \leq \sum_{i=0}^{n} \binom{n}{i} = 2^n.$$

□

It is well-known that all finite languages are regular. Hence we see that many regular languages are not recognized by graph algebra automata. The following example shows that the class of languages recognized by graph algebra automata is not closed under union, intersection, or product.

Example 7 The languages $L_1 = \{x_1x_2, x_2\}$ and $L_2 = \{x_2x_3, x_3\}$ are recognized by graph algebra automata, because they satisfy condition (a) of Theorem 1. However, their union $L_1 \cup L_2$ contains the words x_1x_2, x_2x_3, but does not contain $x_1x_2x_3$. Hence $L_1 \cup L_2$ does not satisfy condition (a). Obviously, it does not satisfy (b) either, because all languages with this condition are infinite. Therefore $L_1 \cup L_2$ is not recognized by a graph algebra automaton.

The languages $\overline{L_1} = X^* \setminus L_1$ and $\overline{L_2} = X^* \setminus L_2$ are recognized by graph algebra automata, as well. But their intersection, which is equal to $\overline{L_1 \cup L_2}$, is not recognized by graph algebra automata, because $L_1 \cup L_2 \notin \mathcal{G}$.

The languages $L_3 = \{x_1\}$ and $L_2 = \{x_2x_3, x_3\}$ satisfy condition (a) of Theorem 1, and so they are recognized by graph algebra automata. However, neither (a) nor (b) holds for their product $L_3 \cdot L_2 = \{x_1x_2x_3, x_1x_3\}$.

For any language L and word $u \in X^*$, let $Lu^{-1} = \{w \in X^* \mid wu \in L\}$ and $u^{-1}L = \{w \in X^* \mid uw \in L\}$. A class $\mathcal{L}$ of languages is said to be *closed under left (right) derivative* if $L \in \mathcal{L}$ implies $u^{-1}L \in \mathcal{L}$ (respectively, $Lu^{-1} \in \mathcal{L}$).

Corollary 8 *The class $\mathcal{G}$ is closed under complement, $\mathcal{G}_a$ is closed under intersections and left derivative, and $\mathcal{G}_b$ is closed under union and right derivative.*

Proof. Equations (3) immideately show that $\mathcal{G}$ is closed under complement. It is routine to verify that $\mathcal{G}_a$ is closed for intersection and left derivative and that $\mathcal{G}_b$ is closed under right derivative.

Now, assume that $L_1, L_2 \in \mathcal{G}_b$. Then $\overline{L_1}, \overline{L_2} \in \mathcal{G}_a$, and therefore $\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2} \in \mathcal{G}_a$, because $\mathcal{G}_a$ is closed under intersection. Hence $L_1 \cup L_2 \in \mathcal{G}_b$, as required. □

Corollary 9 *The classes $\mathcal{G}_a$ and $\mathcal{G}_b$ are closed under the Kleene $*$-operation.*

Proof. Suppose that $L \in \mathcal{G}_a$. First, take any word xyu in L^*, where $x, y \in X, u \in X^*$. We have $xyu \in L^n$, for some $n \geq 1$. Consider the leftmost prefix of this word which is in L. If $x \in L$, then $yu \in L^{n-1} \subseteq L^*$. Further, assume that $xyu_1 \in L$ and $u_2 \in L^{n-1}$, for some factorization $u = u_1u_2$, where $u_1, u_2 \in X^*$. Then $yu_1 \in L$, because $L \in \mathcal{G}_a$. Therefore $yu = yu_1u_2 \in L^*$, again.

Second, take $xu, yxv \in L^*$. If $y \in L$, then clearly $yxu \in L^*$. Hence we may assume that $yxv_1 \in L$ and $v_2 \in L^*$, where $v = v_1v_2$. We get $xu_1 \in L$ and $u_2 \in L^*$, where $u = u_1u_2$. Since $L \in \mathcal{G}_a$, we get $yxu_1 \in L$, and therefore $yxu = yxu_1u_2 \in L^*$, again.

Thus the whole condition (a) of Theorem 1 is satisfied for L^*, and therefore $L^* \in \mathcal{G}_a$.

Now, suppose that $L \in \mathcal{G}_b$. Pick up any word $yu \in L^*$, where $y \in X, u \in X^*$ and $x \in X$. We have $yu_1 \in L$ and $u_2 \in L^*$, for some factorization $u = u_1u_2$. It follows that $xyu_1 \in L$, and so $xyu \in L^*$.

Finally, take $yxu \in L^*$ and $v \in X^*$. If $y \in L$, then obviously $xu \in L^*$. Therefore we may assume that $yxu_1 \in L$ and $u_2 \in L^*$, for some factorization

$u = u_1u_2$, where $u_1, u_2 \in X^*$. Since $L \in \mathcal{G}_b$, it follows that $xu_1 \in L$ or $yxv \in L$. In the former case $xu = xu_1u_2 \in L^*$, and in the latter case $yxv \in L^*$. Thus the whole condition (b) of Theorem 1 is satisfied, and so $L^* \in \mathcal{G}_b$. □

The authors are grateful to two referees for valuable advice and suggestions that helped to improve the first version of this paper.

References

[1] J. Berstel, *Finite automata and rational languages: an introduction*, in "Formal Properties of Finite Automata and Applications", Lect. Notes Comp. Sci. **386**, Springer, New York, 1989, 2–14.

[2] E.W. KISS, R. PÖSCHEL, P. PRÖHLE, *Subvarieties of varieties generated by graph algebras*, Acta Sci. Math. **54** (1990), 57–75.

[3] G. Lallement, "Semigroups and Combinatorial Applications", Wiley, New York, 1979.

[4] G.F. McNulty, C. Shallon, *Inherently nonfinitely based finite algebras*, Lect. Notes Math. **1004** (1983), 206–231.

[5] S. Oates-Williams, *Graphs and universal algebra*, Lect. Notes Math. **884** (1981), 351–354.

[6] S. Oates-Williams, *On the variety generated by Murski's algebra*, Algebra Universalis **18** (1984), 175–177.

[7] G. Paun, A. Salomaa, "New Trends in Formal Languages", Lect. Notes Comp. Sci. 1218, Springer-Verlag, Berlin, 1997.

[8] R. Pöschel, *The equational logic for graph algebras*, Z. Math. Logik Grundlag Math. **35** (1989), no. 3, 273–282.

[9] R. Pöschel, *Graph algebras and graph varieties*, Algebra Universalis **27** (1990), no. 4, 559–577.

[10] R. Pöschel, W.Wessel, *Classes of graphs definable by graph algebra identities or quasi-identities*, Comment. Math. Univ. Carolinae **28** (1987), no. 3, 581–592.

[11] G. Rozenberg, A. Salomaa (Eds.), "Handbook of Formal Languages", Vol. 1, 2, 3, Springer, New York, 1997.

Received November, 2000

Acta Cybernetica **15** (2001) 53-74.

P Systems with Picture Objects

Shankara Narayanan Krishna * Raghavan Rama *

Kamala Krithivasan †

Abstract

New computability models called P systems, based on the evolution of objects in a membrane structure, were recently introduced. In this paper, we consider two variants of P systems having "complex objects" like pictures as the underlying data structure. The first variant is capable of generating pictures with interesting patterns. We also investigate the generative power of this variant by comparing it with the families of two dimensional matrix languages. The second variant has some applications in pattern generation.

1 Introduction

The name 'picture processing' is generally used to describe that area of computer science which is concerned with the analysis and generation of pictures. Pioneering work in suggesting and applying a linguistic model for the solution of nontrivial problems in picture processing was done by Narasimhan [4]. Narasimhan has also proposed and implemented schemes for the recognition of handprinted letters of the English alphabet and for the generation of poster pictures [5, 6].

Various classes of pictures have also been generated using grammars [11]. A matrix model to describe digital pictures viewed as matrices ($m \times n$ rectangular arrays of terminals) is given in [13]. There has been considerable interest in applying the methods of mathematical linguistics to picture generation and description. The concept of substitution of regular sets into languages of Chomsky type are defined in [13] and the concept of substitution of regular sets into L-systems are defined in [7]. When a picture is described as a rectangular array of terminals, it is advantageous to assign attribute values to members of the array, the typical attribute values being intensity or grey level, color and opaqueness or transparency [12].

P systems, introduced by Gh.Paun, [8] form a new class of biologically inspired distributed computing models. P systems can be used as a support for a computing device based on any type of objects and any type of evolution rules associated with them. In basic variants of P systems, the objects are represented either by symbols

*Department of Mathematics Indian Institute of Technology, Madras E-mail:ramar@iitm.ac.in

†Department of Computer Science and Engineering, Indian Institute of Technology, Madras E-mail:kamala@iitm.ernet.in

from a given alphabet or by strings over a given alphabet. In the case of string objects, the objects can evolve in many ways defined by string processing rules [8], [10] such as rewriting (sequential and parallel)[2], point mutations and so on.

Generalizing the passing from symbol objects to string objects, an immediate further step is to consider still more complex objects, such as trees, graphs of arbitrary forms, arrays etc, [9]. Such generalizations are classic in language theory (graph grammars, array grammars and even picture grammars are well developed domains). So, it is natural to start considering P systems with complex object descriptions. With such motivation we introduce picture objects into P systems and evolve the pictures by specific rules.

In the following section, we give a few preliminary notions and notations. In Section 3, a variant of P systems with picture objects is introduced and in the next section, certain examples are given. In Section 5, the generative power of this variant is investigated by comparing it with the existing families of matrix languages $PSML, CSML, CFML$ and RML. In Section 6, it is proved that an online tesselation automata can be simulated using this variant. Another variant of P systems with picture objects is introduced in Section 7, and the application of this variant in pattern generation is illustrated with a few examples.

2 Preliminaries

In this section, we give some preliminaries which will be useful in subsequent sections.

A two dimensional string (or a picture) over an alphabet Σ is a two dimensional rectangular array of elements of Σ. The set of all two dimensional strings over Σ is denoted as Σ^{**}.

Given a picture $p \in \Sigma^{**}$, let $l_1(p)$ denote the number of rows of p and $l_2(p)$ denote the number of columns of p. The pair $(l_1(p), l_2(p))$ is called the size of the picture p. The empty picture is the only picture of size (0,0) and it will be denoted by λ. Pictures of size $(0,n)$ or $(n,0)$ where $n > 0$ are not defined. The set of all pictures over Σ of size (m,n) with $m,n > 0$ will be indicated by $\Sigma^{m \times n}$. Furthermore, if $1 \leq i \leq l_1(p)$ and $1 \leq j \leq l_2(p)$, $p_{i,j}$ denotes the symbol in p with coordinates (i,j).

Now we will give some simple examples of two dimensional languages.

1. Let $\Sigma = \{a\}$ be a one letter alphabet. The set of pictures of a's with three columns is a two dimensional language over Σ. It can be formally defined as $L = \{p \mid p \in \Sigma^{**} \text{ and } l_2(p) = 3\}$.

2. Take a two letter alphabet $\Gamma = \{0,1\}$, and consider the set of squares in which all letters in the main diagonal are 1, whereas the remaining positions carry letter 0. An example is the following one:

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

Two dimensional online tesselation automata ($2OTA$) introduced by Inoue and Nakamura is a particular model of two dimensional cellular automata ($2CA$) [1]. A $2OTA$ is a restricted type of $2CA$ in which cells do not make transitions at every time-step: rather a "transition wave" passes once diagonally across the array. Each cell changes its state depending on the two neighbors to the top and to the left, respectively.

A non-deterministic (deterministic) two-dimensional on-line tesselation automaton, referred to as $2OTA$ ($2DOTA$), is defined by $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, where:
— Σ is the input alphabet;
— Q is a finite set of states;
— $I \subset Q$ ($I = \{i\} \subseteq Q$) is the set of "initial" states;
— $F \subseteq Q$ is the set of "final" (or "accepting") states;
— $\delta : Q \times Q \times \Sigma \to 2^Q$ ($\delta : Q \times Q \times \Sigma \to Q$) is the transition function.

#	#	#	#	#	#	#	#
#							#
#							#
#					$p_{i-1\ j}$		#
#				$p_{i\ j-1}$	$p_{i\ j}$		#
#							#
#	#	#	#	#	#	#	#

A run of $\mathcal{A}$ on a picture p consists of associating a state from Q to each position (i, j) of p. Such a state is given by the transition function δ and depends on the states already associated with positions $(i, j-1)$ and $(i-1, j)$ and on the symbol $p_{i,j}$. At time $t = 0$ an initial state q_0 is associated with all positions of the first row and of the first column of p. The computation consists of $l_1(p) + l_2(p) - 1$ steps. It starts at time $t = 1$ by reading $p_{1,1}$ and associating the state $\delta(q_0, q_0, p_{1,1})$ with position (1,1). At time $t = 2$, states are simultaneously associated with positions (1,2) and (2,1), and so on, to the next diagonals. At time $t = k$, states are simultaneously associated with each position (i, j) such that $i + j - 1 = k$. A $2OTA$ recognizes a picture p if there exists a run of $\mathcal{A}$ on p such that the state assigned to position $(l_1(p), l_2(p))$ is a final state.

Now we give an example of a language recognized by a $2OTA$. Let $\Sigma = \{a\}$ and let $L \subseteq \Sigma^{**}$ be the language of all pictures over Σ with an odd number of columns. That is, $L = \{p \mid l_2(p) \text{ is odd }\}$. A 2 OTA can recognize pictures of L by associating states "1" and "2" with the positions of each odd and even column, respectively. A picture is accepted if positions of the rightmost column contain state "1". More formally, L is recognized by the $2OTA$ $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ defined as follows:

$Q = \{0,1,2\}$, $I = \{0\}$, $F = \{1\}$,
$\delta(0,0,a) = \delta(0,2,a) = \delta(1,0,a) = \delta(1,2,a) = 1$, $\delta(0,1,a) = \delta(2,1,a) = 2$.

A *matrix grammar with appearance checking* is a construct $G = (N,T,S,M,F)$, where N,T are disjoint alphabets, $S \in N$, M, a finite set of sequences of the form $(A_1 \to x_1,\ldots,A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F, a set of occurrences of rules in M (we say that N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w,z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned). We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use grammars without appearance checking, then the family obtained is denoted by MAT. It is known that $MAT \subset MAT_{ac} = RE$ and that each one-letter language in the family MAT is regular.

A matrix grammar $G = (N,T,S,M,F)$ with appearance checking is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S,\#\}$, with these three sets mutually disjoint, and the matrices in M are one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X,Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X,Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

A 2D-matrix grammar is a 2-tuple

$$G = (G_1, G_2)$$

where

$G_1 = (H_1, I_1, P_1, S)$ *is a grammar,*
H_1 *is a finite set of horizontal nonterminals,* $H_1 \cap I_1 = \emptyset$,
$I_1 = \{S_1, S_2, \ldots, S_k\}$, *a finite set of intermediates,*
P_1 *is a finite set of production rules called horizontal production rules,*

S is the start symbol, $S \in H_1$,
$G_2 = (G_{2,1}, G_{2,2}, \ldots, G_{2,k})$,
$G_{2i} = (V_{2i}, I_2, P_{2i}, S_i)$, $1 \leq i \leq k$ are regular grammars,
I_2 is a finite set of terminals,
V_{2i} is a finite set of vertical nonterminals, $V_{2i} \cap V_{2j} = \emptyset, i \neq j$,
S_i is the start symbol,
P_{2i} is a finite set of right linear production rules.

The type of G_1 gives the type of G, so we speak about regular, context-free, context-sensitive, recursively enumerable 2D-matrix grammars if G_1 is a regular, context-free, context sensitive or arbitrary respectively. Derivations are defined as follows: First a string $S_{i_1} S_{i_2} \ldots S_{i_k} \in I_1^*$ is generated horizontally using the horizontal production rules P_1 in G_1. That is, $S \Longrightarrow S_{i_1} S_{i_2} \ldots S_{i_k} \in I_1^*$. Vertical derivations proceed as follows: We write

$$\boxed{A_{i_1} \ldots A_{i_n}} \Downarrow \boxed{\begin{matrix} a_{i1} \ldots a_{in} \\ B_1 \ldots B_n \end{matrix}}$$

if $A_{ij} \rightarrow a_{ij} B_j$ are rules in $P_{2,j}$, $1 \leq j \leq n$. The derivation terminates if $A_j \rightarrow a_{mj}$ are all terminal rules in G_2. The set $M(G)$ of all matrices generated by G is defined to be the set of $m \times n$ *arrays* $[a_{ij}]$ such that $1 \leq i \leq m, 1 \leq j \leq n$ and $S \Longrightarrow_{G_1}^* S_1 \ldots S_n \Longrightarrow_{G_2}^* [a_{ij}]$. Now we shall recall the definition of P systems with string objects from [8]. A rewriting P system of degree $m, m \geq 1$, is a construct

$$\Pi = (V, T, \mu, M_1, M_2, \ldots, M_m, (R_1, \rho_1), (R_2, \rho_2), \ldots, (R_m, \rho_m))$$

where V is the total alphabet, $T \subseteq V$ is the output alphabet, μ is a membrane structure consisting of m membranes labeled with $1, 2, \ldots, m$, $M_1, \ldots, M_m$, are finite languages over V associated with the regions $1, 2, \ldots, m$ of μ, $R_1, \ldots, R_m$ are finite sets of developmental rules over V associated with the regions $1, 2, \ldots, m$ of μ and $\rho_1, \ldots, \rho_m$ are partial order relations over $R_1, \ldots, R_m$, specifying a priority relation among the rules. The rules in R_i are of the form $X \rightarrow (v, tar)$, where $tar \in \{here, out, in_j\}$; j is the label of a membrane. In each step, each string which can be rewritten is rewritten (in a context-free manner) by a rule from its region. The rule is nondeterministically chosen, and the resulting string will be moved to the membrane indicated by the target *tar* associated with the rule: *here* means that the string remains in the same membrane, *out* means that the string exits the membrane, and in_j means that the string goes to the membrane with label j providing that it is directly inside the membrane where the rule $X \rightarrow (v, in_j)$ is applied. This way, we pass from a given configuration to another one; a sequence of transitions forms a computation. We consider as successful computations only the halting ones (computations which reach a configuration where no further rule

can be applied); the result of a halting computation consists of all strings over T^* which are sent out of the system during the computation.

3 Basic definitions

Here, we directly define the system which we are going to work with.
Definition 3.1 A P System of degree n, $n \geq 1$ with picture objects is a construct

$$\Pi = (V, T, \mu, S', M_1, M_2, \ldots, M_n, (R_1, \rho_1), (R_2, \rho_2), \ldots, (R_n, \rho_n)),$$

where

- V is the total alphabet of the system, $V = \bigcup_{i=1}^{n} H_i \cup \bigcup_{i=1}^{n} I_i \cup \bigcup_{i=1}^{n} V_i \cup U \cup T$, where H_j, I_j and V_j are the set of horizontal nonterminals, intermediates and vertical nonterminals associated with membranes j, $1 \leq j \leq n$. U is a set of nonterminals disjoint from any of H_j, I_j, V_j and T.
- $T \subseteq V$ is the output alphabet.
- μ is the membrane structure of degree n, with membranes labeled by $1, 2, \ldots, n$.
- S' is the set of horizontal and vertical scanners. If a membrane has scanners in it, then any picture in the membrane can be scanned. The presence of scanners in a membrane is optional.
- Each M_j is the union of three finite sets, $M_j = S'_j \cup L'_j \cup U_j$, where S'_j is the set of scanners associated with membrane j, L'_j is a finite language over V associated with membrane j and U_j is a multiset of objects over U associated with membrane j.
- R_j and ρ_j are the set of rewriting rules and partial order relations associated with the regions j, $1 \leq j \leq n$, of μ; the form of rules will be specified below.

$M_1, M_2, \ldots, M_n$ can be empty and the same is valid for $R_1, R_2, \ldots, R_n$ and their associated priority relations $\rho_1, \rho_2, \ldots, \rho_n$. Now we shall explain how the scanners work.

The scanners are of two types : horizontal and vertical. The horizontal scanners scan the rows and the vertical scanners scan the columns. Suppose that there are m horizontal scanners $h_1, h_2, \ldots, h_m$ and n vertical scanners $v_1, v_2, \ldots, v_n$ in a membrane k. If p is a picture of size $m \times n$, present in k, then h_i scans the ith row of p and v_j scans the jth column. The horizontal scanner h_i starts scanning the ith row from the left boundary. That is from position $(i, 1)$. Similarly, the vertical scanner v_j starts scanning the jth column from the position $(1, j)$.

At any point of time, the horizontal scanners h_i and vertical scanners v_j hold information about the current element under scan and the previous element scanned. By h_i (or v_j) scanning the ith row (jth column), we mean that h_i (or v_j) is reading

the elements of the ith row (jth column) starting from the left end (top) and proceeds in a systematic way reading one element in each step till it reaches the last element in the ith row (jth column). The scanners $h_1, \ldots, h_m, v_1, \ldots, v_n$ can start working simultaneously unless there are priority relations specifying the order in which they should work.

To specify the position in the picture where the scanning is done, we denote by $h_i(\lambda, p_{i,1})$ (or $v_j(\lambda, p_{1,j})$), the scanners h_i and v_j, during the first step of scanning. This indicates that the current elements under scan are the elements at positions $(i,1)(\ (1,j)\ )$. The scanners h_i, v_j can continue the scanning in the next step, provided there are no rules (horizontal, vertical or transition) having a higher priority. If the scanning is continued in the next step, $h_i(\lambda, p_{i,1})$ $(v_j(\lambda, p_{1,j}))$ is changed to $h_i(p_{i,1}, p_{i,2})$ $(v_j(p_{1,j}, p_{2,j}))$. This means that $p_{i,2}(\ p_{2,j}\ )$ and $p_{i,1}(\ p_{1,j}\ )$ are the current (previous) elements scanned by h_i and v_j respectively.

Since each scanner has information about the current element and its previous one, two operations : *p extract* and *c extract* are defined on them. *p extract* stands for extracting the previous element scanned by the scanner and *c extract* stands for extracting the current element scanned. These are denoted as follows: The operation *p extract* denoted as $h_i^p(p_{i,j}, p_{i,j+1})$ $(v_i^p(p_{j,i}, p_{j+1,i}))$ gives $p_{i,j}$ $(p_{j,i})$, and the operation *c extract* denoted as $h_i^c(p_{i,j}, p_{i,j+1})$ $(v_i^c(p_{j,i}, p_{j+1,i}))$ gives $p_{i\ j+1}$ $(p_{j+1\ i})$. These operations are useful for accessing and changing any particular element in a given picture. For instance, if we want to change the element at position (i,j) in a picture, there are two ways: (1) One way is to use the horizontal scanner h_i and wait till it reaches the jth element. Once it reaches the jth element, $h_i^c(p_{i,j-1}, p_{i,j})$ contains the element at position (i,j). At this step, if an assignment of the form $h_i^c(p_{i,j-1}, p_{i,j}) \to t$ is made, then the element at position (i,j) is changed to t. A similar way is to use the rule $h_i^p(p_{i,j}, p_{i,j+1}) \to t$. (2) Using the vertical scanner v_j also, this can done. Rules of the form $h_i^p(\lambda, p_{i,1}) \to t$ are not valid as there is no element to the left of the first element in any row. Similarly, rules of the form $v_j^p(\lambda, p_{1,j}) \to t$ are also not valid as there are no elements on top of the first element in any column. The work of each scanner h_i (*or* v_j) comes to an end after it has finished scanning all the elements in the ith row (jth column). After this it can start scanning again from the first position of each row (column).

Now we shall explain the three kinds of rules: horizontal, vertical and transition.

Each membrane j has a set I_j of intermediates, horizontal nonterminals H_j and vertical nonterminals V_j and rules associated with each of them. In addition, we also have a set of nonterminals U which can be associated with some or all of the membranes. Also, U is disjoint from all H_j, V_j and I_j. Note that $H_j \cap V_j = \emptyset, H_j \cap I_j = \emptyset$ for all j but it is not necessary that $H_i \cap I_j = \emptyset$ or $H_i \cap V_j = \emptyset$ or $H_i \cap H_j = \emptyset$ or $V_i \cap V_j = \emptyset$ or $I_i \cap V_j = \emptyset$ or $I_i \cap I_j = \emptyset$ for $i \neq j$. Horizontal rules are associated with nonterminals, vertical rules are associated with the intermediates and vertical nonterminals and transition rules are associated with the elements of U. By default, preference is given to the horizontal rules over the vertical rules in each membrane. A string over the intermediates is generated using the horizontal rules after which, the vertical rules are applied. The transition rules are applicable always

unless they are controlled by priority relations. With suitable priority relations, the transition rules can be used to control rules involving scanners; viz., rules of the form $h_i^c(a_{i,j}, a_{i,j+1}) \to t\,(h_i^p(a_{i,j}, a_{i,j+1}) \to t)$ or $v_j^c(a_{i-1,j}, a_{i,j}) \to t\,(v_j^p(a_{i-1,j}, a_{i,j}) \to t)$.

The horizontal rules in each membrane j are of the following two forms:

1. Rules of the form $\alpha \to (\beta_1\gamma_1, tar_1)\dots(\beta_n\gamma_n, tar_n)$ or $\alpha \to (\beta_1, tar_1)\dots(\beta_n, tar_n)$ where $\alpha, \gamma_i \in H_j, \beta_i \in I_j^*$ and tar_i is one of $\{here, out, in_k\}$, k is a membrane adjacent to j. Rules of this form are called right linear rules. The elements of I_j formed during the evolution are terminals with respect to horizontal derivation rules, since only vertical rules are applicable to them.

2. Rules of the form $\alpha \to (\beta_1, tar_1)\dots(\beta_n, tar_n)$ where $\alpha \in H_j$ and $\beta_i \in (H_j \cup I_j)^*$. Rules of this form are called context free rules. As above, tar_i has to be one of *here*, *out* or in_k for some membrane k adjacent to j.

If $n > 1$ in the above rules of types 1, 2, we say that the horizontal rules are replication rules. The horizontal rules are applied sequentially: we apply exactly one rule to a string at a time. If the number of horizontal nonterminals in a string is more than one and if there are applicable rules to each one of them, then a horizontal nonterminal is chosen nondeterministically and one occurrence of it is replaced. The parallelism of the system refers to applying rules to strings in all membranes. If a rule of the form $\alpha \to (\beta\gamma, tar)$ is applied, the string moves as a whole to the membrane indicated by tar after replacing α by $\beta\gamma$. Since preference is given to the horizontal rules in each membrane and each horizontal rule gives rise to some intermediates, a string over intermediates is obtained when the horizontal derivations come to a halt.

The intermediates are the start symbols for vertical rules in all membranes. The vertical rules are applied in parallel: that is, all the intermediates or vertical nonterminals which can be replaced in a step should be replaced. The vertical rules in a membrane j are of the form $\alpha \to (\beta\gamma, tar)/(\beta, tar)$ where $\alpha \in I_j$ or V_j, $\beta \in T^*, \gamma \in V_j$ or I_j and tar is one of $\{here, out, in_k\}$ where k is a membrane adjacent to j. Hence, the vertical rules are always right linear.

The vertical rules result in the string growing downward; that is if there is a string XYZ in a membrane and if there are vertical rules $X \to +++, Y \to .++, Z \to ..+$, then we get the picture

$$\begin{array}{ccc} + & . & . \\ + & + & . \\ + & + & + \end{array}$$

Assume that we rewrite in this way n symbols, by rules which have associated targets of the form $here, out, in_j$ of several types and that there are n_{tar} targets of each type. Then, the structure obtained by rewriting is sent to the membrane indicated by the target with the maximal n_{tar} (the target which appears the maximal number of times in the used rules). As usual, when several targets have the same maximal number of occurrences, then one of them is nondeterministically

chosen. For instance, if we have the structure given below in some membrane k and rules $A_1 \to (\varpi, in_1), A_2 \to (\varsigma, in_2), A_3 \to (\gamma, in_3), A_4 \to (\kappa, in_4)$ where $1, 2, 3, 4$ are membranes adjacent to k, then

$$\begin{array}{cccc} a & A_2 & c & d \\ e & & f & h \\ A_1 & & A_3 & i \\ & & & A_4 \end{array} \quad \Longrightarrow \quad \begin{array}{cccc} a & \varsigma & c & d \\ e & & f & h \\ \varpi & & \gamma & i \\ & & & \kappa \end{array}$$

and the resultant structure can be moved to any of the membranes since each target has occurred an equal number of times.

The priority among rules in any membrane can be summarized as follows: All horizontal rules have higher priority than vertical rules; once all horizontal derivations are over and a string over intermediates is obtained, all symbols in the string which can be rewritten are rewritten according to priority relations that exist between the vertical rules.

In any membrane i, the transition rules for objects of U are of the form $u \to v$, where $u \in U$ and $v = v'$ or $v = v'\delta$, where v' is a string over $(U \times \{here, out\}) \cup (U \times \{in_j \mid 1 \leq j \leq n\})$, j is a membrane adjacent to i and δ is a special symbol not in V. The strings u, v are understood as representations of multisets over U. We use these rules mainly for controlling the action of the scanners. Thus, the horizontal and vertical rules are rewriting rules whereas, the transition rules are evolution rules [8].

Thus, this variant of P systems has a combination of symbol objects described by U and string objects over $H_j, 1 \leq j \leq n$. By applying transition rules to objects of U, horizontal rules to strings over H_j and then vertical rules to objects over $I_j \cup V_j$, pictures are developed. If a picture purely over T is obtained by applying rules of the above type and if it is sent out of the system at the end of a halting configuration, we list it in the language generated by the system. Any picture not over T or any non rectangular array over T sent out of the system are not listed in the language. Similarly, pictures over T remaining in the system after a halting configuration will not be included in the language generated.

If the horizontal rules in all membranes are right linear, then we say that the P system with picture objects uses right linear rules for its horizontal derivations. If the horizontal rules used in some membranes are right linear and the horizontal rules used in some membranes are context free, we say that the P system with picture objects uses context free rules for its horizontal derivations.

The work of this class of systems can be summarized as follows: in each time unit, each string which can be rewritten is rewritten using the horizontal rules; once a string over intermediates is formed, vertical rules are applied simultaneously to all symbols which can be rewritten. In addition, transition rules can be applied to symbols of U present in all the membranes. We thus pass on from one configuration to another one. A sequence of transitions form a computation, and we consider as successful computations the halting ones (the computations which reach a configuration where no rule can be applied); the result of a halting configuration consists of all pictures over T sent out of the system during the computation.

We denote by $P(\Pi)$ the language generated by a P system Π with picture objects; the family of all languages of this type, generated by systems with at most m membranes, using right linear rules (context free rules) for its horizontal derivations, is denoted by $RLP_m(\Pi)(CFP_m(\Pi))$; if no bound on the number of membranes is considered, then the subscript m is replaced by $*$.

4 Examples

Example 4.1 In this example, we generate a four pronged fork without handle, which is known to be generated only by a $CSMG$ [13]. Consider a P system with picture objects, having no priorities and using only right linear rules for its horizontal derivations,

$$\Pi = (V, T, \mu, \lambda, M_1, M_2, M_3, M_4, M_5, (R_1, \phi), (R_2, \phi), (R_3, \phi), (R_4, \phi), (R_5, \phi))$$

with

$$\begin{aligned}
V &= \{S, X, S_1', Y_1, S_1'', Y_2, Y_3, \lambda', \lambda'', S_1, S_2, Y, Y', Y'', V_1', V_2'\} \cup \{\star, .\},\\
T &= \{\star, .\},\\
M_1 &= \{S\},\ M_i = \lambda, i \neq 1,\\
\mu &= [_1\ [_2]_2\ [_3[_4[_5]_5]_4]_3\ ]_1,\\
H_1 &= \{S, X, S_1', Y_1, S_1'', Y_2, Y_3, \lambda', \lambda''\},\\
I_1 &= \{S_1, S_2, Y, Y', Y''\},\ V_1 = \{V_1', V_2'\},\\
H_2 &= \{Y', Y''\}, I_2 = \{\lambda\},\ V_2 = \{\lambda\},\\
H_3 &= \{Y, Y_1\}, I_3 = \{S_2\}, V_3 = \{\lambda\},\\
H_4 &= \{Y', Y_1'\}, I_4 = \{S_2\}, V_4 = \{\lambda\},\\
H_5 &= \{Y''\}, I_5 = \{S_2\}, V_5 = \{\lambda\},\\
R_1 &= \{S \to S_1X,\ X \to YS_1',\ S_1' \to S_1Y_1,\ Y_1 \to Y'S_1'', S_1'' \to S_1Y_2\}\\
&\cup \{Y_2 \to (Y''S_1, in_3),\ Y_3 \to (\lambda, in_2),\ \lambda' \to (\lambda, in_2),\ \lambda'' \to \lambda\}\\
&\quad (horizontal\ rules)\\
&\cup \left\{S_1 \to \begin{matrix}\star\\ V_1'\end{matrix},\ V_1' \to \begin{matrix}\star\\ V_1'\end{matrix},\ S_2 \to \begin{matrix}\star\\ V_2'\end{matrix},\ V_2' \to \begin{matrix}.\\ V_2'\end{matrix}\right\}\\
&\cup \{V_1' \to (\star, out), V_2' \to (., out)\}(vertical\ rules),\\
R_2 &= \{Y' \to (\lambda', out),\ Y'' \to (\lambda'', out)\}(horizontal\ rules),\\
R_3 &= \{Y \to (S_2Y_1, in_4),\ Y_1 \to (Y_3, out),\ Y_1 \to Y\}(horizontal\ rules),\\
R_4 &= \{Y' \to (S_2Y_1', in_5),\ Y_1' \to (Y', out)\}(horizontal\ rules),\\
R_5 &= \{Y'' \to (S_2Y'', out)\}(horizontal\ rules).
\end{aligned}$$

In membrane two, the string $S_1S_2^nS_1S_2^nS_1S_2^nS_1$ is generated as a result of horizontal derivations. This string is passed on to the skin membrane where as a result of vertical derivations, we get the four pronged fork without handle.

A four pronged fork without handle

Example 4.2 Consider the P system with picture objects having no priorities, no scanners and using right linear rules for its horizontal derivations,

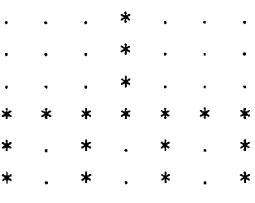
$$\Pi = (V, T, \mu, \lambda, M_1, M_2, \ldots, M_{13}, (R_1, \phi), (R_2, \phi), \ldots (R_{13}, \phi))$$

with

$$\begin{aligned}
V &= \{S, X, Y', Y_2', Y_2'', Y_2''', Y_2^4, Y_2^5, S_1, S_2, S_3, S_1', S_2', S_3', Y'', Y_1'', Y''', Y_1''', Y^4, \\
&\quad Y_1^4, Y^5, Y_1^5, Y^6\}, \\
T &= \{., \star\}, \\
\mu &= [_1\ [_2[_3[_4[_5[_6\ ]_6]_5]_4]_3]_2\ [_7\ ]_7[_8\ ]_8[_9[_{10}[_{11}[_{12}[_{13}\ ]_{13}]_{12}]_{11}]_{10}]_9\ ]_1, \\
M_1 &= \{S\}, M_i = \lambda, i \neq 1, \\
H_1 &= \{S, X, Y', Y_2'\}, I_1 = \{S_1, S_2, S_3, S_1', S_2', S_3'\}, V_1 = \{\lambda\}, \\
H_2 &= \{Y'', Y_1''\}, I_2 = \{S_3\}, V_2 = \{\lambda\}, \\
H_3 &= \{Y''', Y_1'''\}, I_3 = \{S_3\}, V_3 = \{\lambda\}, \\
H_4 &= \{Y^4, Y_1^4\}, I_4 = \{S_3\}, V_4 = \{\lambda\}, \\
H_5 &= \{Y^5, Y_1^5\}, I_5 = \{S_3\}, V_5 = \{\lambda\}, \\
H_6 &= \{Y^6\}, I_6 = V_6 = \{\lambda\}, \\
H_7 &= \{\lambda\}, I_7 = \{S_1, S_2, S_3\}, V_7 = \{S_1', S_2', S_3'\}, \\
H_8 &= \{X, S_3', Y_1'', S_1', Y_1''', S_2', Y_1^4, S_1'', Y_1^5, S_3'', Y_1^6\}, \\
I_8 &= \{Y', S_3, Y'', S_1, Y''', S_2, Y^4, Y^5, Y^6\}, V_8 = \{\lambda\}, \\
H_9 &= \{Y'', Y_2''\}, I_9 = V_9 = \{\lambda\}, \\
H_{10} &= \{Y''', Y_2'''\}, I_{10} = V_{10} = \{\lambda\}, \\
H_{11} &= \{Y^4, Y_2^4\}, I_{11} = V_{11} = \{\lambda\}, \\
H_{12} &= \{Y^5, Y_2^5\}, I_{12} = V_{12} = \{\lambda\}, \\
H_{13} &= \{Y^6\}, I_{13} = V_{13} = \{\lambda\}, \\
R_1 &= \{S \to (S_1 X, in_8),\ Y' \to (S_3 Y_1', in_2),\ Y' \to (Y_2', in_9),\ Y_2' \to \lambda\} \\
&\quad (\textit{horizontal rules}) \\
&\cup \left\{S_1 \to \begin{matrix} \cdot \\ S_1 \end{matrix},\ S_3 \to \begin{matrix} \cdot \\ S_3 \end{matrix},\ S_2 \to \begin{matrix} \star \\ S_2 \end{matrix}\right\} \\
&\cup \left\{S_1 \to \begin{pmatrix} \star \\ S_1 \end{pmatrix}, in_7\right), S_3 \to \left(\begin{pmatrix} \star \\ S_3 \end{pmatrix}, in_7\right), S_2 \to \left(\begin{pmatrix} \star \\ S_2 \end{pmatrix}, in_7\right)\right\}
\end{aligned}$$

$$
\begin{aligned}
&\cup \quad \{S_2' \to (.,out),\ S_3' \to (.,out),\ S_1' \to (\star,out)\}(\textit{vertical rules}),\\
R_2 &= \{Y'' \to (S_3Y_1'',in_3),\ Y_1'' \to (Y'',out)\}(\textit{horizontal rules}),\\
R_3 &= \{Y''' \to (S_3Y_1''',in_4),\ Y_1''' \to (Y''',out)\}(\textit{horizontal rules}),\\
R_4 &= \{Y^4 \to (S_3Y_1^4,in_5),\ Y_1^4 \to (Y^4,out)\}(\textit{horizontal rules}),\\
R_5 &= \{Y^5 \to (S_3Y_1^5,in_6),\ Y_1^5 \to (Y^5,out)\}(\textit{horizontal rules}),\\
R_6 &= \{Y^6 \to (S_3Y^6,out)\}(\textit{horizontal rules}),\\
R_7 &= \left\{S_1 \to \begin{matrix}\star\\ S_1'\end{matrix},\ S_2 \to \begin{matrix}.\\ S_2'\end{matrix},\ S_3 \to \begin{matrix}.\\ S_3'\end{matrix}\right\}\\
&\cup \quad \left\{S_1 \to \begin{pmatrix}\star & , & out\\ S_1' & & \end{pmatrix}, S_2 \to \begin{pmatrix}. & , & out\\ S_2' & & \end{pmatrix}, S_3 \to \begin{pmatrix}. & , & out\\ S_3' & & \end{pmatrix}\right\}\\
&\quad (\textit{vertical rules}),\\
R_8 &= \{X \to Y'S_3',\ S_3' \to S_3Y_1'',\ Y_1'' \to Y''S_1',\ S_1' \to S_1Y_1'''\}\\
&\cup \quad \{Y_1''' \to Y'''S_2', S_2' \to S_2Y_1^4,\ Y_1^4 \to Y^4S_1'',\ S_1'' \to S_1Y_1^5\}\\
&\cup \quad \{Y_1^5 \to Y^5S_3'',\ S_3'' \to S_3Y_1^6,\ Y_1^6 \to (Y^6S_1,out)\}(\textit{horizontal rules}),\\
R_9 &= \{Y'' \to (Y_2'',in_{10}), Y_2'' \to (\lambda,out)\}(\textit{horizontal rules}),\\
R_{10} &= \{Y''' \to (Y_2''',in\ 11), Y_2''' \to (\lambda,out)\}(\textit{horizontal rules}),\\
R_{11} &= \{Y^4 \to (Y_2^4,in_{12}), Y_2^4 \to (\lambda,out)\}(\textit{horizontal rules}),\\
R_{12} &= \{Y^5 \to (Y_2^5,in_{13}), Y_2^5 \to (\lambda,out)\}(\textit{horizontal rules}),\\
R_{13} &= \{Y^6 \to (\lambda,out)\}(\textit{horizontal rule}).
\end{aligned}
$$

This system generates a four pronged fork with handle which cannot be generated even by a $PSMG$, [13]. Initially, we have the start symbol S in the skin. Then, $S_1Y'S_3Y''S_1Y'''S_2Y^4S_1Y^5S_3Y_1^6$ is generated in membrane 8 and in the next step, the string $S_1Y'S_3Y''S_1Y'''S_2Y^4S_1Y^5S_3Y^6S_1$ is sent to the skin by applying the rule $Y_1^6 \to (Y^6S_1,out)$. This string then passes through membranes 2,3,4, 5 and 6 in order, applying rules for $Y, Y', \ldots, Y^6$ and comes back to the skin. This cycle can then be repeated or terminated by applying the rule $Y' \to (Y_2',in_9)$. The string then passes through membranes 9, 10, 11, 12, 13 and comes back to the skin in the form $S_1S_3^{2n+1}S_1S_3^nS_2S_3^nS_1S_3^{2n+1}S_1$. Now vertical rules can be applied suitably to get a four pronged fork with width n in between the prongs.



A four pronged fork with handle

5 The Generative Power

In this section, we compare the power of P systems with picture objects with the families of 2D matrix languages.

Theorem 5.1 *A P system with picture objects, having horizontal scanners h_1, no priorities and using only right linear rules for its horizontal derivations can generate any picture which belongs to the families of $PSML$, $CSML$, $CFML$ and RML.*

Proof. Since all the vertical derivations are right linear by definition, and the highest family of matrix languages is obtained using a $PSMG$ where the horizontal rules used are of type-0, we prove here that a P system with picture objects having horizontal scanners and which uses only right linear rules for its horizontal derivations can generate as a result of its horizontal rules any string belonging to a type-0 language. A P system with picture objects with the above property can then simulate any $PSMG$, the vertical rules of the P system being the same as that of the $PSMG$.

Now we construct a P system with picture objects, having horizontal scanners, no priorities and using only right linear horizontal rules to generate any type-0 language by its horizontal derivations. Let $G = (N, T, S, P)$ be a type-0 grammar in Kuroda normal form. So the rules in G are of the following forms:

(1) $X \to AB, X, A, B \in N$

(2) $XY \to AB, X, Y, A, B \in N$

(3) $X \to a, X \in N, a \in T$

(4) $X \to \lambda, X \in N$

For any $X \in N$, let k be the maximum number of rules of the form $XY \to AB$. For $X \in N$, if there exist nonterminals $X_1, X_2, \ldots, X_k$ in G having rules $XX_i \to \alpha$, $\alpha \in N^+, \mid \alpha \mid = 2$, then we associate to X the membrane structure $\mu_X = [x[x_{1X}]x_{1X}\ [x_{2X}]x_{2X} \cdots [x_{kX}]x_{kX}]x$. Construct the P system with picture objects Π having degree less than or equal to $n(k+1)+2$, $\mid N \mid = n$, given by

$(V, T', \mu, S', M_1, M_{1'}, M_X, M_{X_{1X}}, \ldots, M_{X_{kX}}, \ldots, M_{Z_{kZ}},$
$(R_1, \phi), (R_{1'}, \phi), (R_X, \phi), \ldots, (R_{Z_{kZ}}, \phi))$
with

$V = N \cup T \cup T' \cup \{X', A_X \mid A, X \in N, XY \to AB \in P\},$

$T' = T \cup V_1,$

$\mu = [_1\ \mu_X\ \mu_Y \ldots \mu_Z\ ]_{1'}\ ]_1$, *the number of sub membrane constructs μ_X embedded within μ depends on the number of nonterminals X in the given grammar that have rules of the form $XY \to AB$,*

$S' = \{h_1\},$

$M_1 = \{S\}, M_X = \{h_1 \mid X \in N\},\ M_i = \{\lambda\},\ \forall$ *other* $i,$

$$
\begin{aligned}
H_1 &= \{X \mid X \in N\},\\
I_1 &= \{a \mid a \in T\} \cup \{A' \mid A \in N\} \cup \{A_X \mid XY \to AB \in P, A, X \in N\},\\
V_1 &= \{Any\ set \mid H_1 \cap V_1 = I_1 \cap V_1 = \emptyset\},\\
H_{1'} &= \{A' \mid A \in N\},\ I_{1'} = \{A \mid A \in N\},\ V_{1'} = \{\lambda\},\\
H_X &= H_1 \cup \{A' \mid A \in N\},\ I_X = \{A \mid A \in N\},\ V_X = \{\lambda\},\\
H_{Y_X} &= \{A_X \mid XY \to AB \in P, A, X \in N\},\ I_{Y_X} = \{A' \mid A \in N\},\ V_{Y_X} = \{\lambda\},\\
R_1 &= \{X \to a, \lambda \mid X \to a, \lambda \in P, X \in N, a \in T\}\\
&\cup \{X \to (A'B, in_{1'}) \mid X \to AB \in P\} \cup \{X \to (A_X, in_X) \mid XY \to AB \in P\}\\
&\cup \{X \to X \mid X \in N\ and\ has\ no\ rules\ in P\}(horizontal\ rules)\\
&\cup \{Arbitrary\ vertical\ rules\},\\
R_{1'} &= \{A' \to (A, out) \mid A \in N\},\\
R_X &= \{Y \to (B, in_{Y_X}) \mid h_1^p(an\ element\ of\ V, Y) = A_X\ and\ XY \to AB \in P\}\\
&\cup \{A' \to (A, out) \mid A \in N\},\\
R_{Y_X} &= \{A_X \to (A', out)\}.
\end{aligned}
$$

Initially, the start symbol S is in the skin membrane. Rules in G of the form $X \to a, \lambda$ where $a \in T$ are retained as such in R_1. A rule of the form $X \to AB$ is simulated by applying $X \to (A'B, in_{1'})$ where $A' \in I_1$. This makes the rule right linear with respect to the skin membrane. The rule $A' \to (A, out)$ is then applied to the string in membrane $1'$. Thus the rule $X \to AB$ in G is simulated correctly and the string is back in the skin membrane. To simulate the rule $XY \to AB$, the rule $X \to (A_X, in_X)$ is applied first. In membrane X, we have the horizontal scanner h_1. The rule $Y \to (B, in_{Y_X})$ is now applied only if the element to the left of Y in the string is A_X. The scanner scans the whole string and if there is atleast one configuration in which $h_1^p(a_{1n}, Y)$ equals A_X, that occurrence of Y is replaced and the string moves to membrane Y_X. Here, we convert the A_X into A' and send the string out. Back in membrane X, the rule $A' \to (A, out)$ is applied and the string goes back to the skin membrane after simulating the rule $XY \to AB$ correctly. Proceeding in this way, we get a string over I_1 in the skin membrane (Corresponding to getting a string over T in G). Now any vertical rule can be applied to generate a picture. Thus, any picture generated by the matrix languages $PSML, CSML, CFML$ and RML can be generated by a P system using horizontal scanners and right linear rules for its horizontal derivations. □

Theorem 5.2 *A P system with picture objects, no scanners, no priorities and using context free replicated rules for its horizontal derivations can generate any picture belonging to the families of $PSML, CSML, CFML$ and RML.*

Proof. It is proved in [3] that any string can be generated by a replicated rewriting P system with no priorities. In proving this result we considered a matrix grammar $G = (N, T, S, M, F)$ in the binary normal form with matrices $m_1, \ldots, m_k$ of type 2 or 4 and $m_{k+1}, \ldots, m_l$ of type 3 and constructed a P system having the membrane structure $\mu = [_1\ [_2\ ]_2 \ldots [_k\ ]_k\ [_{k+1}]_{k+1} \ldots [_l\ ]_l\ ]_1$.

Here, we construct a P system with picture objects having the membrane structure $[_0\ \mu\ ]_0$, with $H_0 = H_1 = N, H_i = N \cup \{X_i \mid X \in N, k+1 \leq i \leq l\}, H_i = N \mid 1 \leq i \leq k, V_j = \emptyset, j \neq 0, I_j = T, \forall j$, where T and N are respectively the set of terminals and non terminals of the given grammar G. The set V_0 can be chosen to be disjoint from H_0 and I_0 and the vertical rules in the skin membrane can be chosen arbitrarily. $M_1 = \{S\}$, the start symbol of G, $M_i = \emptyset, i \neq 1$. We introduce the horizontal rules $S \to XA$ in membrane one and $\{A \to A \mid A \in N\}$ in the skin membrane; the horizontal rules in the rest of membranes are the same as in [3]. The strings reaching membrane zero are the strings which were sent out of the system in the above lemma. If a string which is not purely over T^* (here I_1^*) reaches membrane zero, the rule $A \to A$ is applied for all nonterminals A occurring in the string and hence the computation never stops. Only when a string over I_1^* reaches the skin membrane, can the vertical rules be applied. In this way, any picture belonging to the families of $PSML, CSML, CFML$ and RML can be generated. □

Theorem 5.3 *A P system with picture objects having priorities for horizontal rules, no scanners and using context free rules for horizontal derivations can generate any picture belonging to the families of $PSML, CSML, CFML$ and RML with just 3 membranes.*

Proof. We recall the result $RE = ERP_2(Pri)$ proved in [2]. In proving this result we consider a matrix grammar $G = (N, T, S, M, F)$ in the binary normal form with matrices $m_1, \ldots, m_k$ of type 2 or 4 and $m_{k+1}, \ldots, m_l$ of type 3 and construct a P system having the membrane structure $\mu = [_1\ [_2\ ]_2\ ]_1$.

Here, we construct a P system with picture objects having the membrane structure $[_0 \mu]_0$, $H_0 = H_1 = H_2 = N$, $I_0 = I_1 = I_2 = T$, $V_0 =$ any set disjoint from H_0 and I_0, $V_1 = V_2 = \{\lambda\}, M_0 = M_2 = \{\lambda\}, M_1 = \{S\}$, *the start symbol of* G. The horizontal rules and priorities for membranes one and two are the same rules and priorities used in the proof of the theorem $RE = ERP_2(Pri)$. As in the above theorem, we add rules $\{A \to A \mid A \in H_0\}$ for all nonterminals in the skin membrane. This ensures that vertical derivations start only from a string over I_0^*. The vertical rules may be chosen arbitrarily. This way, any picture in the families of $PSML, CSML, CFML$ and RML can be generated by a 3-degree P system with picture objects. □

6 P Systems and OTA

In this section we simulate the action of an OTA using P systems with picture objects. We prove that given a $2DOTA$ recognizing or not recognizing a given picture, there exists a P system with picture objects that works exactly in the same way as the $2DOTA$.

Theorem 6.1 *Given a $2DOTA$ recognizing or not recognizing a picture, we can*

construct a two degree P system with picture objects having scanners which can simulate the 2DOTA.

Proof. Let $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ be a $2DOTA$ and let p be a picture of size $m \times n$. $\mathcal{A}$ recognizes p if there exists a run on p such that the state associated to the position (m, n) is a final state.
We construct a two degree P system with picture objects which simulates $\mathcal{A}$.
Let

$$\Pi = (V, \{t\}, S', [_1\ [_2\ ]_2\ ]_1, \{\lambda\}, \{p, c_0, h_1, h_2, \ldots, h_m, v_1, v_2, \ldots, v_n\}, (R_1, \phi), (R_2, \rho_2))$$

where p is the given picture of size $m \times n$ for which transitions of the $2DOTA$ are defined. $h_1, \ldots, h_m$ and $v_1, \ldots, v_n$ are the horizontal and vertical scanners in membrane two, which depend on the size of the given picture p.

$$\begin{aligned}
V &= \Sigma \cup Q \cup \{\gamma_1, \gamma_2, t_1, d, \dagger\} \cup \{d_q \mid q \in Q\} \cup \{c_i \mid 0 \le i \le m+n\} \cup \{t, t_1\},\\
U &= \{\dagger, \gamma_1, \gamma_2, d, t, t_1\} \cup \{d_q, c_i \mid q \in Q, 0 \le i \le m+n\},\\
S' &= \{h_1, h_2, \ldots, h_m, v_1, v_2, \ldots, v_n\},\\
H_1 &= I_1 = V_1 = \{\lambda\},\\
H_2 &= I_2 = V_2 = \{\lambda\},\\
R_1 &= \{t_1 \to (t, out)\}(\textit{transition rule}),\\
R_2 &= \{c_0 \to c_1\gamma_1, c_i \to c_{i+1}, 1 \le i \le m+n-2, c_{m+n-2} \to c_{m+n-1}d\}\\
&\cup \{\gamma_1 \to \gamma_2, \gamma_2 \to \gamma_1, \dagger \to \dagger, d \to d_{\delta(v_n^p(p_{m-1,n}, p_{m,n}),\ h_m^p(p_{m,n-1}, p_{m,n}), p_{m,n})}\}\\
&\cup \{d_q \to t_1\delta \mid q \in F\} \cup \{d_q \to \dagger \mid q \notin F\}(\textit{transition rules}).
\end{aligned}$$

For $1 \le j \le m+n-1,\ 1 \le i \le m$, scanner rules:

$$h_i^c(p_{i,j-1}, p_{i,j}) \to \begin{cases}
\delta(v_j^p(p_{i-1,j}, p_{i,j}),\ h_i^p(p_{i,j-1}, p_{i,j}), p_{i,j}) & \text{if } v_j^p(p_{i-1,j}, p_{i,j}),\ h_i^p(p_{i,j-1}, p_{i,j}) \ne \lambda\\
\delta(q_0,\ h_i^p(p_{i,j-1}, p_{i,j}), p_{i,j}) & \text{if } v_j^p(p_{i-1,j}, p_{i,j}) = \lambda, h_i^p(p_{i,j-1}, p_{i,j}) \ne \lambda\\
\delta(v_j^p(p_{i-1,j}, p_{i,j}), q_0, p_{i,j}) & \text{if } h_i^p(p_{i,j-1}, p_{i,j}) = \lambda, v_j^p(p_{i-1,j}, p_{i,j}) \ne \lambda\\
\delta(q_0, q_0, p_{i,j}) & \text{otherwise}
\end{cases}$$

$$\begin{aligned}
\rho_2 = \{&h_{i-1} > h_i, v_{i-1} > v_i\ \forall i;\ \gamma_1 \to \gamma_2 > \textit{all scanning rules};\\
&\gamma_2 \to \gamma_1 > \begin{cases} d \to d_{\delta(\ v_n^p(p_{m-1,n},\ p_{m,n}),\ h_m^p(p_{m,n-1},\ p_{m,n}),\ p_{m,n})},\\ \textit{all rules with LHS } c_j \textit{ and } h_i^c(p_{i,l-1}, p_{i,l}) \end{cases}\\
&c_i \to c_{i+1} > \textit{all rules with LHS } h_j^c(p_{j,l-1}, p_{j,l}) \textit{ such that } i+1 \ne j+l\}.
\end{aligned}$$

We start the proof by giving the details of the rules applied during the first few steps.
Step 0 :We have an $m \times n$ picture and a counter c_0 in membrane two.
Step 1 : the scanners h_1 and v_1 start working and $c_0 \to c_1\gamma_1$ is applied. So we have in the second membrane $h_1(\lambda, p_{1,1})$ and $v_1(\lambda, p_{1,1})$ in addition to c_1 and γ_1.
Step 2 : the rules $\gamma_1 \to \gamma_2$, $c_1 \to c_2$ and $h_1^c(\lambda, p_{1,1}) \to \delta(q_0, q_0, p_{1,1})$ are applied. In

this step, we assign to $p_{1,1}$ the state $\delta(q_0, q_0, p_{1,1})$ of the $2DOTA$ by applying the rule $h_1^c(\lambda, p_{1,1}) \to \delta(q_0, q_0, p_{1,1})$.
Step 3 : $\gamma_2 \to \gamma_1$ and scanning rules $h_1(p_{1,1},\ p_{1,2})$, $v_1(p_{1,1},\ p_{2,1})$, $h_2(\lambda, p_{2,1})$, $v_2(\lambda, p_{1,2})$ are applied.
Step 4 : $\gamma_1 \to \gamma_2$, $c_2 \to c_3$, $h_1^c(p_{1,1}, p_{1,2}) \to \delta(q_0, h_1^p(p_{1,1}, p_{1,2}))$ and $h_2^c(\lambda, p_{2,1}) \to \delta(v_1^p(p_{1,1}, p_{2,1}), q_0, p_{2,1})$ are applied. In this step, states are assigned to the next diagonal elements $p_{1,2}$ and $p_{2,1}$ depending on $h_1^p(p_{1,1}, p_{1,2})$ and $v_1^p(p_{1,1}, p_{2,1})$. That is, depending on the state stored at the position to the left of (top of) $p_{1,2}$ and $p_{2,1}$.
Clearly, this is the same way the $2DOTA$ works; assigning states to diagonal elements depending on the states of the elements at the top and left.

The rules $\gamma_1 \to \gamma_2$ and $\gamma_2 \to \gamma_1$ characterize the assigning phase and scanning phase respectively. If the rule $\gamma_1 \to \gamma_2$ is applied in a step, then values are assigned to the current elements scanned. No scanning is possible in this step due to the priorities. Similarly, when $\gamma_2 \to \gamma_1$ is applied, no values can be assigned to any of the elements; only the scanning takes place. The scanners are activated in the order $h_1, h_2, \ldots$ and $v_1, v_2, \ldots$. Due to this, at each step the elements under scan are the elements falling along a diagonal and the previous elements scanned by h_i and v_j are respectively the elements to the left and top of the currently scanned elements. This helps in assigning states to the currently scanned elements by referring to the states at the top and left positions.

In the kth assigning phase, we apply $c_k \to c_{k+1}$ and assign states to positions (i, j) for which $i + j - 1 = k$. Note that the priority $c_k \to c_{k+1} >$ all rules with LHS $h_i^c(p_{i,j-1}, p_{i,j})$ for which $i + j \neq k + 1$ ensures that at each step only the elements along the diagonal are assigned values. Proceeding in this manner, we assign to the element at the (m, n)th position the state $\delta(v_n^p(p_{m-1,n}, p_{m,n}), h_m^p(p_{m-1,n}, p_{m,n}))$. If this happens to be a final state, the rule $d_q \to t_1\delta$ is applied and the membrane dissolves. In the next step, the rule $t_1 \to (t, out)$ is applied. Thus, if the given picture is recognized by the $2DOTA$, a t will be sent out and the system halts. Otherwise, the rule $d_q \to \dagger$ is applied and the computations never halt. □

7 A Variant of P Systems with Picture Objects

In this section, we consider a variant of P systems with picture objects. Operations like reflection, rotation, magnification on an $m \times n$ picture produce another picture. To achieve the effect of such operations on a picture using membranes, we define a new class of P systems with picture objects. Because of the form of the basic operation it uses, we call such a system a block-rewriting P system. As we will see, this variant is useful for exemplifying the power of membrane structures. Formally, we give the definition as follows:

Definition 7.1 *A block-rewriting P system with picture objects of degree n is a*

construct

$$\Pi = (V, T, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n)$$

where

- V *is the total alphabet of the system;*
- T *is the output alphabet;*
- μ *is the membrane structure;*
- $M_1, \ldots, M_n$ *are sets of pictures initially present in the regions* $1, \ldots, n$ *of* μ*;*
- $R_i, 1 \leq i \leq n$ *are rules associated with the regions* $1, \ldots, n$ *of* μ*;*

The objects considered here are pictures of the form $\begin{matrix} \# & \# & \ldots \# & \# \\ \# & a_{11} & \ldots a_{1n} & \# \\ \# & a_{21} & \ldots a_{2n} & \# \\ \vdots & \vdots & \ldots \vdots & \vdots \\ \# & a_{m1} & \ldots a_{mn} & \# \\ \# & \# & \ldots \# & \# \end{matrix}$. Rules are applied to sub pictures of any given picture. We consider a sub picture of a picture to be a rectangular block of the form $\boxed{\begin{matrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{matrix}}$ or $\boxed{p_1 \ldots p_n}$, $p_i \in (V \cup \{\#\})^*$, of size $1 \times n$ or $n \times 1$. Any block which contains the boundary marker $\#$ at one or both its ends is called an end block. We apply rules to blocks of size $1 \times n$ or $n \times 1$ replacing them by blocks of size $1 \times m$ or $m \times 1$, $m \geq n$. For blocks which are not end blocks, the replacement is made with a block of the same size, and for end blocks the replacement is done with blocks of the same or higher dimension. This is to take care of the shearing effect which occurs otherwise. For an end block, the increase in length of the resultant block does not affect the positions of any of the symbols in the picture. For instance, consider the block $\boxed{12 \ldots n\#}$ and the rule $12 \ldots n\# \rightarrow a_1 \ldots a_{n+1}\#\#$. On applying this rule, the block $\boxed{12 \ldots n\#}$ in the picture is replaced by $\boxed{a_1 a_2 \ldots a_{n+1}}$ and $\#\#$ is placed next to that, increasing the column size of the picture. Since the increase was made at the end, the positions of all other symbols in the picture are not distorted. Similar is the case when applying such rules to $\boxed{\begin{matrix} \# \\ 1 \\ \vdots \\ n \end{matrix}}$ or $\boxed{\begin{matrix} 1 \\ \vdots \\ n \\ \# \end{matrix}}$ or $\boxed{\#12 \ldots n}$. As in the case of rewriting P systems, we can attach targets to the rules. The rules are applied in a maximally parallel manner, that is, we replace all blocks in the picture for which there are rules available in the given region. If any two rules involve overlapping blocks

say for instance, $\boxed{\#a_{i,1}a_{i,2}\dots a_{i,n}}$ and $\boxed{\begin{matrix} a_{1,1} \\ \vdots \\ a_{i,1} \\ \vdots \\ a_{m,1} \end{matrix}}$ then we choose one of the blocks nondeterministically and apply the rule. Assume that we rewrite in this way n blocks, by rules which have associated targets of the form $here, out, in_j$ of several types and that there are n_{tar} targets of each type. Then, the picture obtained by rewriting is sent to the membrane indicated by the target with the maximal n_{tar} (the target which appears the maximal number of times in the used rules). The result of a computation consists of the set of pictures over T^* sent out of the skin membrane at the end of a halting configuration. We illustrate the effectiveness of this system with a few examples.

Example 7.1 *Consider the P system*

$$\Pi = (V, T, [_1[_2[_3[_4[_5\ ]_5]_4]_3]_2]_1, M_1, \dots, M_5, R_1, \dots, R_5)$$

with

$$\begin{aligned}
V &= \{a, b, c, d, \#, \#', \#'', \#_1, \#_2, \#_a, \#_b, \#_c, \#_b\}, \\
T &= \{a, b, c, d, \#\}, \\
M_1 &= \begin{matrix} \#'' & \# & \# & \#' \\ \# & a & b & \# \\ \# & c & d & \# \\ \#'' & \# & \# & \#' \end{matrix}, M_i = \lambda, i \neq 1, \\
R_1 &= \{\#\#' \to (\#\#\#_1, in_2), b\# \to (bb\#_b, in_2), d\# \to (dd\#_d, in_2)\} \\
&\cup \{\#' \to (\#, out), \#'' \to (\#, out), \#_b \to \#, \#_d \to \#\}, \\
R_2 &= \left\{ \begin{matrix} c \\ \# \end{matrix} \to \left(\begin{matrix} c \\ c \\ \#_c \end{matrix}, in_3 \right), \begin{matrix} d \\ \# \end{matrix} \to \left(\begin{matrix} d \\ d \\ \#_d \end{matrix}, in_3 \right) \right\} \\
&\cup \left\{ \begin{matrix} \# \\ \#'' \end{matrix} \to \left(\begin{matrix} \# \\ \# \\ \#_2 \end{matrix}, in_3 \right), \begin{matrix} \#_d \\ \#_1 \end{matrix} \to \left(\begin{matrix} \#_d \\ \#_d \\ \#_1 \end{matrix}, in_3 \right) \right\} \\
&\cup \{\#_a \to (\#, out), \#_c \to (\#, out)\}, \\
R_3 &= \{\#a \to (\#_a aa, in_4), \#c \to (\#_c cc, in_4), \#''\# \to (\#_2\#\#, in_4)\} \\
&\cup \{\#_2\#_c \to (\#_3\#_c\#_c, in_4), \#_3 \to (\#_3, out), \begin{matrix} \#_c \\ \#_3 \end{matrix} \to \left(\begin{matrix} \#_c \\ \#'' \end{matrix}, out \right)\}, \\
R_4 &= \left\{ \begin{matrix} \# \\ a \end{matrix} \to \left(\begin{matrix} \#_a \\ a \\ a \end{matrix}, in_5 \right), \begin{matrix} \# \\ b \end{matrix} \to \left(\begin{matrix} \#_b \\ b \\ b \end{matrix}, in_5 \right) \right\}
\end{aligned}$$

$$\cup \left\{ \begin{matrix} \#_2 \\ \#_a \end{matrix} \rightarrow \begin{pmatrix} \#_3 \\ \#_a \\ \#_a \end{pmatrix}, in_5 \right\}, \begin{matrix} \#_1 \\ \#_b \end{matrix} \rightarrow \begin{pmatrix} \#_4 \\ \#_b \\ \#_b \end{pmatrix}, in_5 \Bigg) \Bigg\}$$

$$\cup \left\{ \begin{matrix} \#_3 \\ \#_a \end{matrix} \rightarrow \begin{pmatrix} \#'' \\ \#_a \end{pmatrix}, out \right\},$$

$$R_5 = \{\#_d\#_1 \rightarrow (\#_d\#', out), \#_b\#_4 \rightarrow (\#_b\#', out)\}.$$

Clearly, this system scales up the given picture (excluding the boundary marker #) by factors of 2. Hence, the system generates all pictures of dimension $2^n \times 2^n, n \geq 1$ over a, b, c, d.

Example 7.2 *Consider the P system*

$$\Pi = (V, T, [_1[_2\ ]_2]_1, M_1, M_2, R_1, R_2)$$

$$V = \{., \star, \#, \#', \$\},$$

$$T = \{., \star, \#, \$\},$$

$$M_1 = \lambda,\ M_2 = \begin{matrix} \# & \# & \# & \# & \# & \#' \\ \$ & . & \star & . & . & \$ \\ \$ & . & \star & . & . & \$ \\ \$ & . & \star & . & . & \$ \\ \$ & . & \star & \star & \star & \$ \\ \# & \# & \# & \# & \# & \#' \end{matrix},$$

$$R_1 = \{.\$ \rightarrow (..\$, out), .\$ \rightarrow (..\$, in_2), \star\$ \rightarrow (\star\star\$, out), \star\$ \rightarrow (\star\star\$, in_2)\}$$

$$\cup \{\#\#' \rightarrow (\#\#\#, out), \#\#' \rightarrow (\#\#\#', in_2)\},$$

$$R_2 = \left\{ \begin{matrix} . \\ \# \end{matrix} \rightarrow \begin{pmatrix} . \\ . \\ \# \end{pmatrix}, out \right), \begin{matrix} \$ \\ \#' \end{matrix} \rightarrow \begin{pmatrix} \$ \\ \$ \\ \#' \end{pmatrix}, out \Bigg) \Bigg\}$$

$$\cup \left\{ \begin{matrix} \$ \\ \# \end{matrix} \rightarrow \begin{pmatrix} \$ \\ \$ \\ \# \end{pmatrix}, out \right), \begin{matrix} . \\ \star \\ \# \end{matrix} \rightarrow \begin{pmatrix} . \\ . \\ \star \\ \# \end{pmatrix}, out \Bigg) \Bigg\}$$

$$\cup \left\{ \begin{matrix} \star \\ \star \\ \# \end{matrix} \rightarrow \begin{pmatrix} \star \\ \star \\ \star \\ \# \end{pmatrix}, out \right) \Bigg\}.$$

Initially, membrane two contains a picture with an 'L' shape engraved in it. The system sends out pictures after enlarging the 'L' in both directions. Thus this system is capable of producing pictures with L's of any size engraved in it.

Example 7.3 *Consider the P system*

$$\Pi = (V, T, [_3[_1[_2\ ]_2]_1]_3, M_1, \lambda, \lambda, R_1, R_2, R_3)$$

with

$$
\begin{aligned}
V &= \{\$, \$', \$'', \#, a, a', a'', b, b', b'', c, c', c''\},\\
T &= \{\$, \#, a, b, c\},\\
M_1 &= \begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \$ & \# & a & b & c & \# \\ \$ & \# & b & c & a & \# \\ \$ & \# & c & a & b & \# \\ \# & \# & \# & \# & \# & \# \end{array},\\
R_1 &= \{\$\# \to \#\$, \{\$x \to \$'\$^x \mid x \in \{a,b,c\}\} \cup \{\$^x y \to y\$^x \mid x, y \in \{a,b,c\}\}\\
&\cup \{\$^x \# \to (x'\#, in_2) \mid x \in \{a,b,c\}\} \cup \{\$^y x'' \to (y'x'', in_2) \mid x, y \in \{a,b,c\}\}\\
&\cup \{\$'x'' \to x''\$'' \mid x \in \{a,b,c\}\} \cup \{\$''x'' \to x''\$'' \mid x \in \{a,b,c\}\}\\
&\cup \{\$''x \to \$'\$^x \mid x \in \{a,b,c\}\} \cup \{\$''x' \to x'\$'' \mid x \in \{a,b,c\}\}\\
&\cup \{\$''\# \to (\#\$, out)\},\\
R_2 &= \{\{yx' \to \$^y x'' \mid x, y \in \{a,b,c\}\} \cup \{x\$^y \to \$^y x \mid x, y \in \{a,b,c\}\}\\
&\cup \{x''\$^y \to (x''y, out) \mid x, y \in \{a,b,c\}\} \cup \{\$'x' \to (\$'x'', out) \mid x \in \{a,b,c\}\}\\
&\cup \{\$'\$^x \to (\$'x'', out) \mid x \in \{a,b,c\}\}\},\\
R_3 &= \{\{x'' \to (x, out) \mid x \in \{a,b,c\}\}\}.
\end{aligned}
$$

Given a picture or a set of pictures in membrane one, the system outputs those pictures which are obtained from the initial ones by taking a vertical reflection. In our case, because we start from $\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \$ & \# & a & b & c & \# \\ \$ & \# & b & c & a & \# \\ \$ & \# & c & a & b & \# \\ \# & \# & \# & \# & \# & \# \end{array}$, *the system outputs the picture* $\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & c & b & a & \# & \$ \\ \# & a & c & b & \# & \$ \\ \# & b & a & c & \# & \$ \\ \# & \# & \# & \# & \# & \# \end{array}$.

8 Conclusion

We have introduced a variant of P systems having pictures as the data structure. It has been shown that this variant is able to generate the family $PSML$ with no priorities, scanners and no bound on the number of membranes. The family $PSML$ can also be generated by two other variants of this system: one with replicated rewriting as the underlying control structure with no bound on the number of membranes, another one having three membranes and using rewriting and priorities among the rules. A different type of P systems with picture objects is defined in section 7, which is motivated by the idea of simulating an operation on a picture using membrane structures. A further study of this variant is worthwhile.

References

[1] K.Inoue and A.Nakamura, Some properties of two-dimensional on-line tesselation acceptors, *Information Sciences*, Vol. 13, 1977, 95–121.

[2] S.N.Krishna and R.Rama, On Power of P systems based on Sequential and Parallel Rewriting, *Intern. J. Comp. Math.*, Vol.76, 1–2, 2000, 317–330.

[3] S.N.Krishna, R.Rama P Systems with Replicated Rewriting, *Journal of Automata, Languages and Combinatorics*, to appear.

[4] R.Narasimhan, Labeling schemate and syntactic description of pictures, *Information and control*, 7, 1964, 151–179.

[5] R. Narasimhan,On the description, generation and recognition of classes of pictures, *In Automatic interpretation and classification of images*, Academic Press, 1969.

[6] R.Narasimhan and V.S.N.Reddy, A generative model for handprinted English letters and its computer implementation, *ICC Bulletin*, 6, 1967, 275–287.

[7] N.Nirmal and R.Rama, Picture generation and developmental matrix systems, *Computer Vision, Graphics and Image Processing*, Vol 43, pp 67-80, 1988.

[8] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[9] Gh.P ăun, Computing with P Systems: Twenty Six Research Topics, *Auckland University, CDMTCS Report* No 119, 2000 (www.cs.auckland.ac.nz/CDMTCS).

[10] Gh.Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266, (www.tucs.fi).

[11] A.Rosenfeld, *Picture Languages*, Academic Press, New York, 1979.

[12] G.Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Hiedelberg, 1997.

[13] G.Siromoney, R.Siromoney and K.Krithivasan, Abstract families of matrices and picture languages, *Computer Graphics and Image Processing*, (1972), 1, (284-307)

Received January, 2001

Acta Cybernetica **15** (2001) 75-100.

Logical definability of Y-tree and trellis systolic ω-languages*

Monti Angelo† and Peron Adriano‡

Abstract

In this paper we investigate the correspondence (in the style of the well known Büchi Theorem) between ω-languages accepted by systolic automata and suitable (proper) extensions of the Monadic Second Order theory of one successor ($MSO[<]$). To this purpose we extend Y-tree and trellis systolic automata to deal with ω-words and we study the expressiveness, closure and decidability properties of the two classes of ω-languages accepted by Y-tree and trellis automata, respectively. We define, then, two extensions of $MSO[<]$, $MSO[<, adj]$ and $MSO[<, 2x]$, which allow to express Y-tree ω-languages and trellis ω-languages, respectively.

1 Introduction

The subject of automata accepting infinite sequences was established in the sixties by Büchi, McNaughton and Rabin (for a survey, see [16, 17]). One motivation for considering automata on infinite sequences (Büchi automata) was the analysis of the sequential calculus ($MSO[<]$), a system of monadic second order logic for the formalisation of properties of sequences. Büchi showed that any condition on sequences that it is written in this calculus can be reformulated as a statement about acceptance of sequences by a Büchi automaton (Büchi Theorem). The resulting theory is fundamental for those areas in computer science where non-terminating computations are studied, for instance modal logics of programs and specification and verification of concurrent systems (e.g. see [15]). This paper is an attempt to set a correspondence between ω-languages accepted by systolic automata and suitable (proper) extensions of $MSO[<]$. Systolic automata (see [8] for a survey) are synchronous networks of (memoryless) processors working in discrete time. These automata have been the main theoretical models used to study various basic problems concerning systolic architectures, systems and computations. In this paper we

*Research partially supported by MURST Progetto Cofinanziato TOSCA.

†Dipartimento Scienze dell'Informazione, Università di Roma (La Sapienza), 00198, Via Salaria 113, Italy, e-mail: monti@dsi.uniroma1.it

‡Dipartimento di Matematica e Informatica, Università di Udine, 33100, Via Zanon 6, Italy, e-mail: peron@dimi.uniud.it

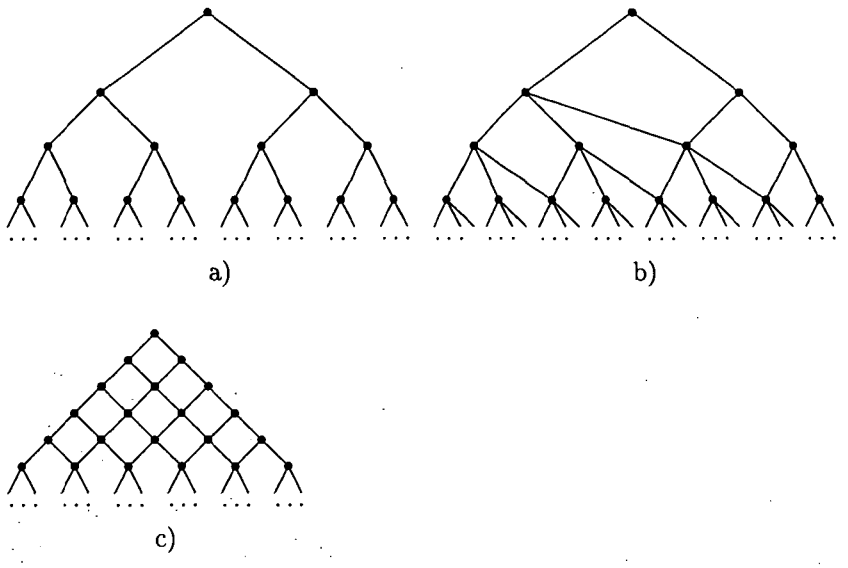


Figure 1: Interconnection structures for systolic automata: binary tree (a), Y-tree (b) and trellis (c).

consider systolic automata enforcing three of the most widely studied types of interconnection structures, i.e. leafless binary tree, leafless binary Y-tree and downward unbounded trellis (see figure 1). These automata have been mainly considered as acceptors of finitary languages. Only recently, binary tree systolic automata have been considered as acceptors of ω-languages in [12], showing that the class of *binary tree ω-languages* properly extends the class of ω-languages accepted by Büchi automata though preserving the same closure and decidability properties. In this paper we introduce also the class of *Y-tree ω-languages* (i.e. the class of ω-languages accepted by Y-tree), which extends the class of binary tree ω-languages, and the class of *trellis ω-languages* which, in its turn, extends the class of Y-tree ω-languages. The class of Y-tree ω-languages is closed under union and intersection, it is not closed under complement and the emptiness problem is undecidable. The class of trellis ω-languages is closed under union and intersection. The emptiness problem is undecidable. As well as in the finitary case (see [9]), the closure under complementation remains an open question. The logical characterisation of binary tree ω-languages has been established in [13], where it has been shown that such a class of languages corresponds (in the sense of Büchi theorem) to a proper extension of $MSO[<]$ by a suitable function $flip$ (the extension is denoted by $MSO[<, flip]$).

In this paper we propose two suitable extensions of $MSO[<]$, denoted by $EMSO[<, adj]$ (the fragment of $MSO[<, adj]$ allowing only existential quantification over predicates) and $MSO[<, 2x]$, which allow to express Y-tree and trellis ω-languages, respectively. Actually, we believe that $EMSO[<, adj]$ characterises the class of Y-tree ω-languages. The question, whether $MSO[<, 2x]$ is a char-

acterisation of ω-languages accepted by trellis automata, is an (hard) problem directly related to the open problem of the closure under complement of both finitary and ω-languages. The hierarchy of extensions of regular ω-languages accepted by Büchi automata and the corresponding hierarchy of extensions of $MSO[<]$ are summarised in figure 2.

We believe that investigating logical characterisations of systolic languages is meaningful for both theoretical and practical reasons. The results of the characterisation of binary tree ω-languages have been fruitfully applied in [10] for studying the decidability properties of logics for time granularity interpreted over ω-layered metric temporal structures (the traditional basic engine of $MSO[<]$ was, in that case, not powerful enough). We believe that the results presented in this paper could be fruitfully applied for formally studying and comparing the expressive power of meaningful extensions of the temporal logics for time granularity considered in [10]. Moreover, we expect that our investigation could contribute in an original way to the research oriented to developing systematic and sufficiently automatisable methods to synthesise systolic systems from high level specifications.

2 Preliminaries

Throughout this paper Σ denotes an alphabet; $\Sigma^\star$ denotes the set of (finite length) words on Σ, and Σ^ω denotes the set $\omega - words$ on Σ. Finite words are indicated by $u, v, w, \ldots$, ϵ is the empty word and letters $\alpha, \beta, \ldots$ are used for ω-words. The symbol $\cdot$ denotes concatenation on strings. For an ω-word α, $\alpha(i)$, with $i \in \mathbb{N}$, denotes the i-th element of α; $\alpha(m,n)$ denotes the subword $\alpha(m) \cdot \ldots \cdot \alpha(n)$ of α, and $\alpha(n,\omega)$ denotes the suffix $\alpha(n) \cdot \alpha(n+1) \cdot \ldots$ of α. For $V \subseteq \Sigma^\star$, V^ω is the set of ω-words having the form $v_0 \cdot v_1 \cdot v_2 \cdot \ldots$ with $v_i \in V$, for $i \in \mathbb{N}$.

2.1 Logical definability

Properties of words will be described by monadic second order logic formulas. Since logical formulas will be interpreted over words, it is more convenient to identify a word $\alpha \in \Sigma^\omega$ with the structure $\underline{\alpha} = \langle \mathbb{N}, <, (Q_a)_{a \in \Sigma} \rangle$, where $<$ is the usual ordering of natural numbers and $Q_a = \{i : \alpha(i) = a\}$.

In the sequel we shall consider a second order language over a set of (interpreted) binary relation symbols $B_1, \ldots, B_n$, denoted as $MSO[B_1, \ldots, B_n]$ (for notation we follow [17]), whose syntax is as follows.

Given an alphabet of *individual variables* $x, y, z, \ldots$, *predicates* $X, Y, Z, \ldots$ and predicate symbols Q_a (with $a \in \Sigma$),
Atomic formulas have the form $x = y$, xB_iy, $X(x)$ and $Q_a(x)$;
Formulas are built up from atomic formulas by means of the boolean connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and the quantifiers $\forall$ and $\exists$ ranging over both individual variables and predicates.

$$\begin{array}{ccc} \mathcal{L}_\omega(Trellis\ A.) & \subseteq & MSO[<, 2x] \\ \bigcup & & \bigcup | \\ \mathcal{L}_\omega(Y-Tree\ A.) & \subseteq & EMSO[<, adj] \\ \bigcup & & \bigcup \\ \mathcal{L}_\omega(B-Tree\ A.) & = & EMSO[<, flip] = MSO[<, flip] \\ \bigcup & & \bigcup \\ \mathcal{L}_\omega(Buchi\ A.) & = & EMSO[<] = MSO[<] \end{array}$$

Figure 2: The hierarchies of systolic ω-languages and of the related $MSO[<]$ extensions.

Individual variables are interpreted as elements of $\mathbb{N}$ (i.e., positions of an ω-word) and predicates as subsets of $\mathbb{N}$ (i.e., sets of positions of an ω-word). Atomic formulas $x = y$ and $X(x)$ are interpreted as equality between x and y, and $x \in X$. For a fixed interpretation of binary relational symbols B_i (as subset of $\mathbb{N} \times \mathbb{N}$), atomic formulas xB_iy, are interpreted as $\langle x, y\rangle \in B_i$. Given an ω-word α, $Q_a(x)$ is interpreted as $x \in \{i : \alpha(i) = a\}$. Boolean connectives and quantification operators are endowed with the standard semantics.

If $\phi(X_1, \ldots, X_n)$ is a formula with at most the predicates $X_1, \ldots, X_n$ occurring free in ϕ, α is a ω-word and $P_1, \ldots, P_n$ are subsets of $\mathbb{N}$, then we write $\langle \underline{\alpha}, P_1, \ldots, P_n\rangle \models \phi(X_1, \ldots, X_n)$ if α satisfies ϕ under the above mentioned interpretation taking P_i as interpretation of X_i. A formula without free variables and predicates is said to be a *sentence*. If ϕ is a sentence, for sake of simplicity, we write $\underline{\alpha} \models \phi$. The language $L(\phi)$ defined by a sentence ϕ is the set of all words $\alpha \in \Sigma^\omega$ such that $\underline{\alpha} \models \phi$. A language L is $MSO[B_1, \ldots, B_n]$-definable if there is a formula ϕ of $MSO[B_1, \ldots, B_n]$ such that $L = L(\phi)$; L is $EMSO[B_1, \ldots, B_n]$-definable if there is a sentence ϕ having the form $\exists X_1 \ldots \exists X_n \psi(X_1, \ldots, X_n)$ of $MSO[B_1, \ldots, B_n]$ such that $L = L(\phi)$ and ψ contains only first order quantifiers (i.e. quantifiers occurring in ψ range only over individual variables). In the following the binary relational symbol $<$ is interpreted as the standard ordering of natural numbers. In this setting Büchi Theorem can be formulated as follows.

Büchi Theorem An ω-language is regular iff it is $MSO[<]$- definable.

In the sequel, for simplicity, atomic formulas over relational symbols interpreted as functions will be written in functional form (i.e. $flip(x) = y$, $adj(x) = y$, $2x = y$ instead of $x\ flip\ y$, $x\ adj\ y$, $x\ 2\ y$).

3 Systolic computations

Let us recall how systolic automata process a word in the finitary case. Systolic automata are networks of (memoryless) processors (called also nodes in the following) working in discrete time. As far as systolic binary tree are concerned, the

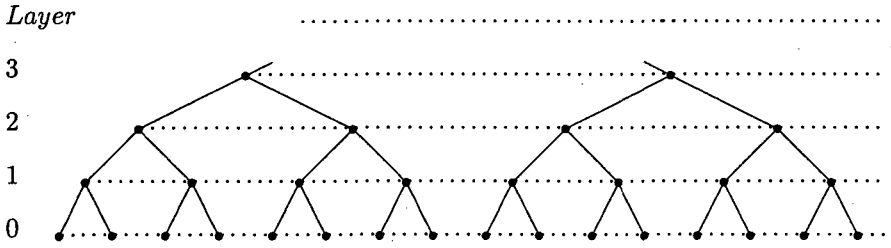


Figure 3: The upward unbounded binary tree structure.

interconnection structure of a binary tree automaton is an (infinite) leafless perfectly balanced binary tree (see figure 1.a). In order to process a word w, whose length $|w|$ is equal to 2^i, the i-th level of the tree is chosen (the level of the root is 0). Now, the automaton is fed in such a way that adjacent processors at level i-th are fed with adjacent symbols of w, and that the leftmost processor is fed with the first symbol of w. All the processors at level i-th synchronously output a symbol belonging to the *state alphabet* Q, according to the *input relation* of the automaton. Each processor at level $(i-1)$-th receives the two states output by its two sons (placed at level i) and it synchronously (with respect to processors at the same level) outputs a symbol belonging to Q according to the *transition relation* of the automaton. Therefore, information flows bottom-up, in parallel and synchronously, level by level. The result of the computation is collected at the root of the tree and the word w is accepted if the state associated with the root is a *final state*.

Let us consider now the case of ω-languages. Though the local topology of the interconnections of processors remains unchanged, the whole structure is generalised. In the finitary case the structure is a leafless binary tree, namely a downward unbounded multilevel structure (provided with infinitely many layers) having a bounded number of processors at each layer. On the contrary, the structures we shall considering are upward unbounded multilevel structures where each layer is a structure isomorphic to natural numbers (with the usual order). In this respect, an upward unbounded structure (see figure 3) can be represented as a grid of processors which is unbounded both in the horizontal dimension (processors in a layer) and vertical dimension (number of layers). In the following we shall mention the i-th column of the structure intending the set of i-th elements of each layer.

Nodes of a layer (with the exception of the 0-th layer) are connected to nodes of the previous layer drawing a regular topology: either binary tree like as in figure 3, or Y-tree like as in figure 6 or trellis like as in figure 13. In such a framework, the ω-word to be processed is associated with the 0-th layer of the upward unbounded structure, adjacent symbols being associated with adjacent nodes. The information flows up, in parallel and synchronously, level by level exactly as in the finitary case. Therefore, a systolic computation is a labelling (over the alphabet of states of the automaton) of the upward unbounded structure compatible with the transition

relation of the automaton. In order to accept/discard a computation on an ω-word, we impose a Büchi criterion on the labels of the 0-th column, namely the computation is successful if an infinite number of final states label that column. In the following we shall see how a computation on a ω-word α, defined here as a labelling of the whole structure, can be indeed defined incrementally step by step as the result of composing the results of finite computations on segments of finite length of α. Notice that the interconnection structure we are proposing allows one to easily simulate the systolic computation defined in the literature for the finitary case.

In the remaining part of this section we recall (from [13]) the definitions and results concerning systolic automata operating over an upward unbounded binary tree structure. The cases of Y-tree and trellis interconnection structures are introduced and discussed in the two next sections.

Definition 1 *A* binary tree automaton *is a tuple* $\mathcal{A} = \langle \Sigma, Q, Q_0, g, f \rangle$, *where*

- Q *is the finite set of* states *and* $Q_0 \subseteq Q$ *is the set of* final states;
- $g \subseteq \Sigma \times Q$ *is the* input relation;
- $f \subseteq Q \times Q \times Q$ *is the* transition relation.

Let us introduce now a notion of step-wise binary tree computation on ω-words. Consider an ω-word α. At each computation step, the automaton process a segment of α whose length is a power of two and doubles step by step. The computation of a systolic automaton $\mathcal{A}$ is defined for a word $w \in \Sigma^*$ only if $|w| = 2^k$ (with $k \geq 0$) and it is given by the relation $O_{\mathcal{A}} \subseteq \Sigma^* \times Q$ defined recursively on k as follows:

- if $|w| = 2^0$, then $\langle w, q \rangle \in O_{\mathcal{A}}$ if and onlyif $\langle w, q \rangle \in g$;
- if $|w| = 2^k$, with $k > 0$, then $\langle w, q \rangle \in O_{\mathcal{A}}$ if and only if $\langle q_1, q_2, q \rangle \in f$, with $\langle w_1, q_1 \rangle, \langle w_2, q_2 \rangle \in O_{\mathcal{A}}$, $|w_1| = |w_2| = 2^{k-1}$ and $w_1 \cdot w_2 = w$.

The step-wise computation of an automaton $\mathcal{A}$ over an ω-word α is recorded into an ω-sequence of states called *B-run*. Such a sequence stores at the i-th position the state q resulting from processing the prefix $\alpha(0, 2^i - 1)$ of α. The state resulting from processing the next prefix $\alpha(0, 2^{i+1} - 1)$ is obtained from q and from the states q', output by the systolic automaton fed with $\alpha(2^i, 2^{i+1} - 1)$, according to the transition relation f. The structure of the step-wise computation is depicted in figure

4. Notice that the B-run is the sequence of states labelling the 0-th column of the structure.

More formally, a *B-run* of $\mathcal{A}$ on $\alpha \in \Sigma^\omega$ is a map $\psi : \mathbb{N} \mapsto Q$ s.t.

- $\langle \alpha(0), \psi(0) \rangle \in g$;
- $\langle \psi(i-1), q, \psi(i) \rangle \in f$, with $\langle \alpha(2^{i-1}, 2^i - 1), q \rangle \in O_{\mathcal{A}}$, for $i > 0$.

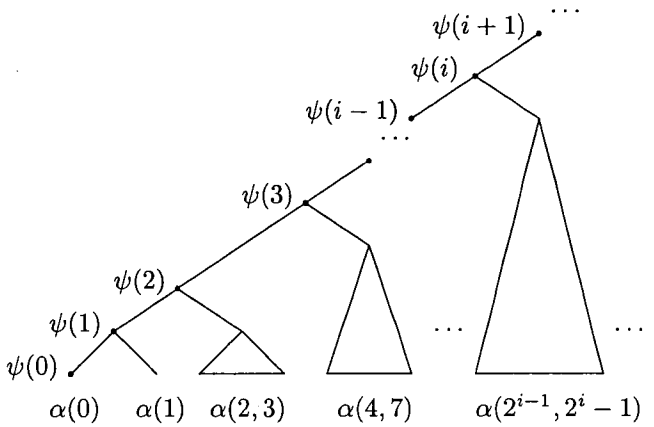


Figure 4: A B-run ψ on α.

A B-run is *successful* iff there are infinitely many i such that $\psi(i) \in Q_0$. An ω-word α is *accepted by* $\mathcal{A}$ iff there exists a successful run of $\mathcal{A}$ on α. The set of ω-words (i.e. *the language*) *accepted by* $\mathcal{A}$ is denoted by $\mathcal{L}_\omega(\mathcal{A})$.

In [13] we proved that the logical counterpart of ω-languages accepted by binary tree automata is obtained by extending $MSO[<]$ with a function *flip* given as follows.

Definition 2 *The function* $flip : \mathbb{N}^+ \to \mathbb{N}$ *is such that*

$$\text{if } x = 2^{k_n} + 2^{k_{n-1}} + \ldots + 2^{k_0} \text{ with } k_n > \ldots > k_1 > k_0, \text{ then } flip(x) = x - 2^{k_0}.$$

The logical definability of ω-languages accepted by binary tree automata is proved by showing that the function $flip$ allows to impose on natural numbers the upward unbound binary tree structure depicted in figure 5. Such a structure is obtained by associating with each even number $z = 2^{k_n} + \ldots + 2^{k_1} + 2^{k_0}$ (with $k_n > \ldots > k_1 > k_0 > 0$), the left and right son, respectively,

$$x = 2^{k_n} + \ldots + 2^{k_1} + 2^{k_0 - 1} \text{ and} \tag{1}$$

$$y = 2^{k_n} + \ldots + 2^{k_1} + 2^{k_0} + 2^{k_0 - 1}. \tag{2}$$

Such a childhood relationship can be easily defined into $MSO[<, flip]$ since

$$x = max\{s : s < z, flip(s) = flip(z)\} \text{ and} \tag{3}$$

$$y = max\{s : flip(s) = z\}. \tag{4}$$

Notice that the elements of the i-th layer (with $i \geq 0$) are

$$\{2^i + k2^{i+1} : k \geq 0\} \tag{5}$$

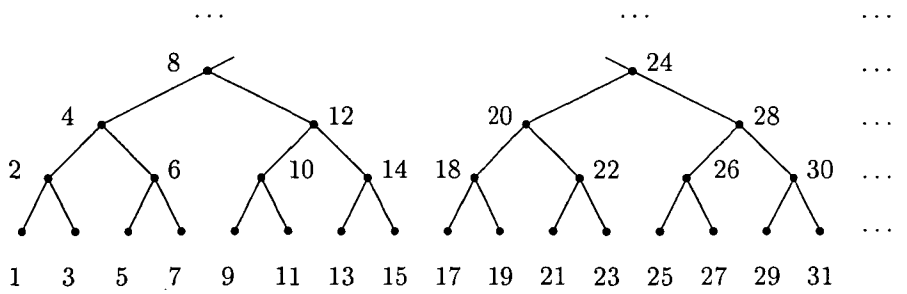


Figure 5: The upward unbounded binary tree structure on $\mathbb{N}$.

and the elements of the i-th column (with $i \geq 0$) are

$$\{(2i+1)2^k : k \geq 0\}. \tag{6}$$

Remark. Notice that the 0-th layer of the upward unbounded binary structure on $\mathbb{N}$ is isomorphic but not equal to $\mathbb{N}$ (even numbers are missing). As a consequence, when such a structure is exploited for simulating a systolic computation on an input word α, α cannot be associated directly with the 0-th layer (i.e. $\alpha(0)$ with 1, $\alpha(1)$ with 3 and so on) since, in this case, the set of numbers $\mathbb{N}$ would not be in correspondence with the index set of α. The correspondence between $\mathbb{N}$ and the index set of α is recovered by associating with the 0-th layer, instead of α, the sequences of states of the systolic automaton $q_1, q_3, q_5, \ldots, q_{2n+1}, \ldots$, with $q_i \in f(g(\alpha(i-1)), g(\alpha(i)))$ (a preprocessing of α).

Theorem 1 *(cf.[13]) An ω-language is accepted by a binary tree automaton iff it is $MSO[<, flip]$-definable (or equivalently, iff it is $EMSO[<, flip]$ -definable).*

4 Y-Tree ω-Languages

The interconnection structure of a Y-tree automaton is a leafless perfectly balanced binary tree augmented with links from a node to the immediate neighbour (if any) of its right son called the *adoptive* son (see figure 1.b). The expressive power and properties of Y-tree automata as acceptors of finitary languages have been studied in [5, 7]. In this section we introduce and study Y-tree automata as acceptors of ω-languages. In this respect, a Y-tree automaton processes a word exactly as a binary tree automaton does with the only exception that the computation is performed over an upward unbounded Y-tree structure (see figure 6).

Definition 3 *A* Y-tree automaton *is a tuple $\mathcal{Y} = \langle \Sigma, Q, Q_0, g, f \rangle$, where*

- *Q is the finite set of* states *and $Q_0 \subseteq Q$ is the set of* final states*;*
- *$g \subseteq \Sigma \times Q$ is the* input relation*;*

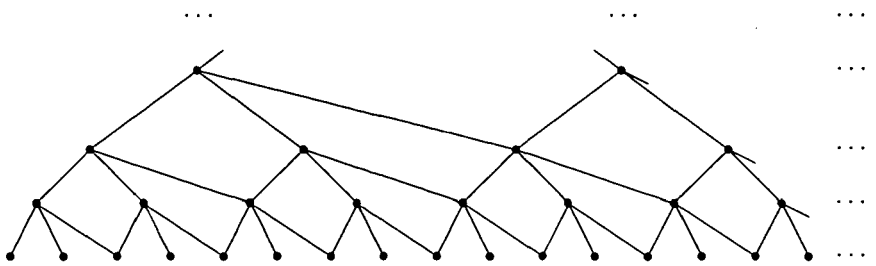


Figure 6: The upward unbounded Y-tree structure.

- $f \subseteq Q \times Q \times Q \times Q$ *is the* transition relation.

A computation of a Y-tree automaton is then a labelling of an upward unbounded Y-tree structure compatible with the input and transition relations of the automaton. As in the case of binary tree automata, we define now the step-wise version of such a computation. Given an ω-word α, at each computation step, the automaton processes a segment of α whose length is a power of two and which doubles step by step. In this way, at the i-th computation step, the prefix $\alpha(0, 2^{i+1}-2)$ of α turns out to be processed, and the result of such a computation is stored into a sequence $q_0 \ldots . q_i$ of states of Q. The sequence $q_0 \ldots . q_i$ is precisely the sequence of states labelling the path of adoptive edges from the i-th element of the 0-th column of the upward unbounded Y-tree structure leading to the 0-th layer (see, for instance, the path labelled by $v(0)v(1)v(2)$ in figure 7 which codifies the result of the 2-th step of a computation). Now, the $i+1$-th step transforms the sequence $q_0 \ldots . q_i$, encoding the computation at the i-th step, into a sequence of states $q'_0 \ldots . q'_i, q'_{i+1}$ as a result of composing $q_0 \ldots . q_i$ with the result of processing the segment $\alpha(2^{i+1}-1, 2^{i+2}-2)$ of α. Notice that, in order to store the information about the computation on the prefix of a ω-word, a sequence (of unbounded length) of states is required (in the case of binary tree automata a single state suffices). For a Y-tree automaton $\mathcal{Y}\langle \Sigma, Q, Q_0, g, f \rangle$, the transformation above, called *run step*, is given by the relation

$$O_{\mathcal{Y}} \subseteq Q^\star \times \Sigma^\star \times Q^\star$$

which satisfies the property that $\langle v, w, u \rangle \in O_{\mathcal{Y}}$ implies $|v| = k$, $|w|2^k$ and $|u| = k+1$, for some $k \geq 0$, and is defined recursively on k as follows (see figure 7 for a graphical hint):

- if $k = 0$, then $\langle \epsilon, w, q \rangle \in O_{\mathcal{Y}}$, with $\langle w, q \rangle \in g$;
- if $k > 0$ and $w_1 \cdot w_2 = w$ with $|w_1| = |w_2| = 2^{k-1}$, then $\langle v, w, q \cdot u_2 \rangle \in O_{\mathcal{Y}}$, where
  - $\langle v(1, k-1), w_1, u_1 \rangle \in O_{\mathcal{Y}}$, $\langle u_1(1, k-1), w_2, u_2 \rangle \in O_{\mathcal{Y}}$, $\langle v(0), u_1(0), u_2(0), q \rangle \in f$.

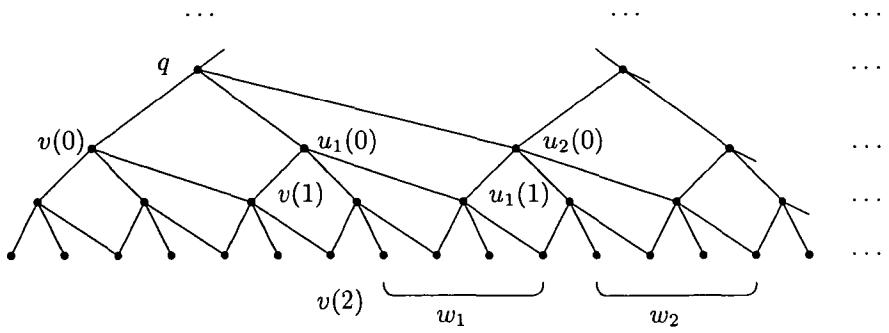


Figure 7: A Y-tree step.

A *Y-run* for an ω-word α stores the result of each computation step and it is defined as a map $\psi : \mathbb{N} \mapsto Q^\star$ such that:

- $\langle \alpha(0), \psi(0) \rangle \in g$;
- $\langle \psi(i), \alpha(2^{i+1} - 1, 2^{i+2} - 2), \psi(i+1) \rangle \in O_{\mathcal{Y}}$, for $i \geq 0$.

A Y-run ψ over $\alpha \in \Sigma^\omega$ is *successful* iff there are infinitely many $i \in \mathbb{N}$ s.t. $\psi(i) = v_i$ and $v_i(0) \in Q_0$. The structure of a Y-run is shown in figure 8. The definition of a ω-language accepted by Y-tree automaton is exactly as the one for binary tree automata.

Example 1 *An instance of Y-tree ω-language is:*

$$L = \{\alpha \in \{0,1\}^\omega : \alpha(2^s + k2^{s+1}) = 1 \text{ for some } s \geq 0, \text{ for all } k \geq 0\}. \quad (7)$$

Notice that $\alpha \in L$ iff the positions where α is set to 1 include at least the set of natural numbers belonging to a layer of the upward unbounded binary tree structure on $\mathbb{N}$ (see figure 5 and Eq.5).
Before giving a formal definition of the automaton accepting the language L, we sketch the idea behind the recognising process. The automaton propagates the symbols $\{0,1\}$, received at its input layer, till a layer which is supposed to be uniformly labelled by symbol 1 is reached. At this point, all of the processors in the active layer check the uniformity of the labelling of the previous layer, by verifying that all of their left and adoptive sons are uniformly labelled, and propagates the successful/unsuccessful result of the test. A symbol is propagated along the adoptive edge at the first step and along the right edge in all the remaining steps. At each step a processor chooses non-deterministically whether to propagate the signal it is receiving from its right son or to perform the uniformity check. The check can be successful only if all of the processors of a layer choose simultaneously and non-deterministically to perform the uniformity check. The automaton $\mathcal{Y} = \langle \Sigma, Q, Q_0, g, f \rangle$ accepting L is given by the following components:

- $Q = \{0, 1, \overline{0}, \overline{1}, ok\}$, $Q_0 = \{ok\}$, $g = \{\langle x, \overline{x} \rangle : x \in \{0,1\}\}$;

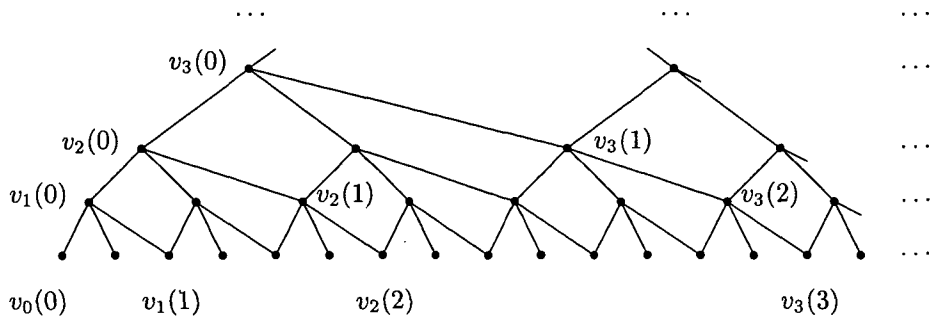


Figure 8: The structure of a Y-run ($v_i = \psi(i)$).

- $f = \{\langle \overline{x}, \overline{1}, \overline{y}, ok\rangle, \langle \overline{x}, \overline{y}, \overline{z}, z\rangle, \langle 1, x, 1, ok\rangle, \langle x, y, z, y\rangle, \langle ok, ok, ok, ok\rangle :$ *with* $x, y, z \in \{0, 1\}\}$.

Over-lining of states is used to distinguish the first step of the computation from the others. The state ok is output after a successful uniformity check. The first and second schemas of tuples manage check and transmission, respectively, at the first step. The third and fourth schemas of tuples manage check and transmission, respectively, in all the other steps.

As far as expressivity is concerned, the class of Y-tree ω-languages includes the class of binary tree ω-languages as an immediate consequence of the fact that the upward unbounded Y-tree structure is a upward unbounded binary tree structure enriched with adoptive edges. In fact, as in the finitary case, Y-tree automata are more expressive than binary tree ones.

Theorem 2 *The class of Y-tree ω-languages strictly includes the class of binary tree ω-languages.*

Proof. We show that the Y-tree language L of Example 1 is not accepted by any systolic binary tree automaton. The proof is by contradiction. Let us assume that there exists a systolic binary tree automaton $\mathcal{A}$ having n states and accepting L. We take now the $n+1$ (different) ω-words $\alpha_0, \ldots, \alpha_n$ of L such that $\alpha_i(j) = 1$ iff $j = 2^i + k2^{i+1}$ (for all $k \geq 0$). The positions set to 1 in α_i are exactly those associated with the i-th layer of the upward unbounded structure. Let ψ_i be a successful B-run of $\mathcal{A}$ for α_i (with $0 \leq i \leq n$). Since the number of states is n, there must be two runs ψ_j and ψ_k with $j \neq k$ such that $\psi_j(n+1) = \psi_k(n+1)$. This immediately implies, by the definition of B-run, that the ω-word

$$\alpha' = \alpha_j(0, 2^{n+1} - 1) \cdot \alpha_k(2^{n+1}, \omega)$$

has a succesful run (e.g. $\psi_j(0, n+1) \cdot \psi_k(n+2, \omega)$). This leads to a contradiction since $\alpha' \notin L$. □

By exploiting standard techniques it is easy to prove the closure properties of Y-tree ω-languages stated in the following proposition.

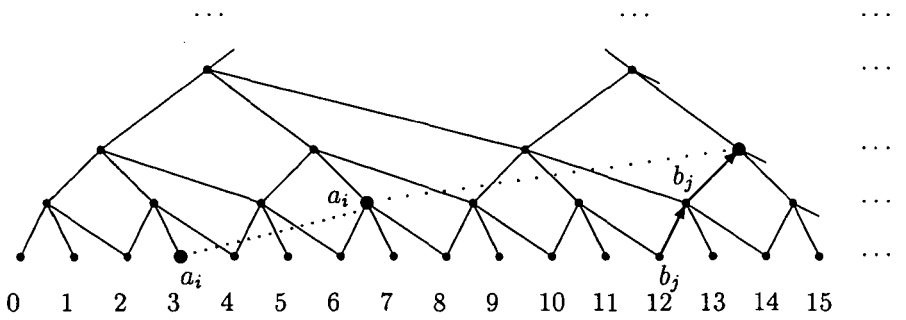


Figure 9: Trasmitting a_i along the third column and b_j along left edges.

Proposition 3 *The class of Y-tree ω-languages is closed under union, intersection and projection.*

As in the finitary case (see [7]), the class of languages accepted by Y-tree automata is not closed under complementation. We show in the following example a Y-tree language whose complement is not accepted by any Y-tree automaton.

Example 2 *An instance of Y-tree ω-language is*

$$L = \{\alpha \in \{0,1\}^\omega : \alpha(2^s + k2^{s+1}) \neq \alpha(2^{s'} + k2^{s'+1}) \text{ for some } s', s, k, \geq 0\}. \quad (8)$$

Notice that $\alpha \in L$ iff there are two positions i and j of α such that $\alpha(i) \neq \alpha(j)$ and i and j belong to the same column of the upward unbounded binary tree structure on $\mathbb{N}$ (see Eq.6). In order to prove that $\alpha \in L$, it is sufficient to check that there exists an odd position i (i.e. the least element of a column) of α such that $\alpha(i) \neq \alpha(j)$ and i and j are elements of the same column. We sketch the algorithm used by the automaton to check that positions i and j as above belong to the same column. The idea is that of (upward) propagating the symbol a_i from position i, and b_j from position j, till they meet at a common node (see figure 9). The symbol b_j is forced to move only along left edges. The symbol a_i is propagated, by exploiting non-determinism, along the i-th column. Following this procedure, symbols a_i and b_j meet if and only if i and j belong to the same column. In fact, if i and j belong to the same column, then $j = i2^k$ (for some k). When j moves from the 0-th layer to the 1-th along a left edge, it reaches the $i2^{k-1}$-th position of that layer, and then the $i2^{k-2}$ at the next step, and so on (see figure 9). It remains to show that it is possible to move the symbol a_i along the i-th column. Let us consider the example of figure 10. The symbol a is input at position 5. At the second computation step, the node at position 5-th in the 1-th layer guesses that it is an element of the 5-th column and it enters the state A. The correctness of such a guess has to be checked. To this purpose, both the nodes labelled by a and A transmit upward a check-symbol (c_a and C_A, respectively). The guess is correct if the sequences of edge types (left/right) in the two paths from the nodes labelled by a and A to the 0-th column is the same. Since the path length could be arbitrarily long and the

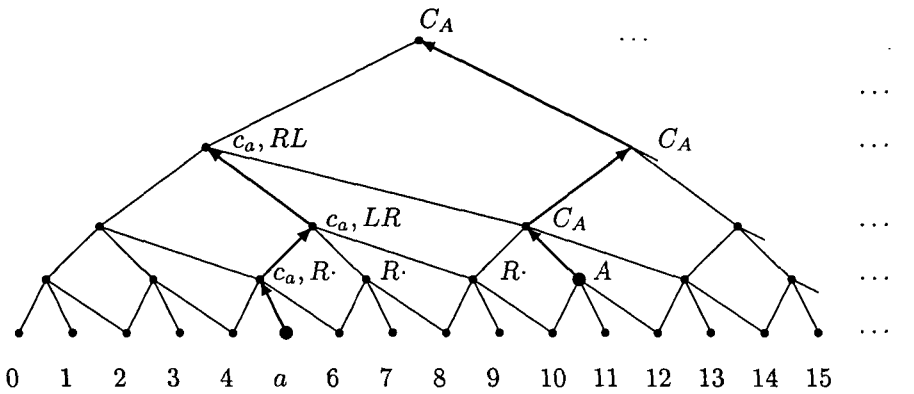


Figure 10: Transmitting a symbol along the 5-th column.

number the states is finite, this comparison cannot be performed at the completion of the path but must be performed step by step in the following way. Each sequence of nodes in the segment of the layer bounded by the path of c_a and C_A keeps trace of the last two types of edges crossed by c_a in its way. In figure 10, $R\cdot$, LR and RL stand for Right, Left – Right and Right – Left, respectively. This information is used by the nodes on the path of C_A to force the crossing of an edge of the same type as the one crossed by c_a a step before. The formal definition of the automaton is now easy and it is, for the sake of simplicity, omitted.

Theorem 4 *The class of Y-tree ω-languages is not closed under complementation.*

Proof. We show that the language

$$L = \{\alpha \in \{0,1\}^\omega : \alpha(2^s + k2^{s+1})\alpha(2^{s'} + k2^{s'+1}),\ \text{for all } s', s \geq 0 \text{ with } k \geq 0\} \quad (9)$$

is not a Y-tree ω-language. Note that such a language is the complement of the Y-tree ω-language of Example 2, since $\alpha \in L$ iff all of the positions of α belonging to the same column are labelled by the same symbol. The proof is by contradiction. Let us assume that there exists a systolic Y-tree automaton $\mathcal{Y}$ having n states and accepting L. For a suitable m, we take now $s = 2^{\frac{2^{m+1}-2}{2}} > n^{m+1}$ (different) ω-words $\alpha_1, \ldots, \alpha_s$ of L such that $\alpha_i(k) \neq \alpha_j(k')$ for odd k, k', with $1 \leq k, k' \leq 2^{m+1} - 2$, for all $1 \leq i < j \leq s$. Notice that each odd number identifies a column, and then α_i and α_j disagree at least in the labelling of a column. Let ψ_i be a succesful Y-run of $\mathcal{Y}$ for α_i (with $1 \leq i \leq s$). Since there are at most $n^{m+1} < s$ different strings on states of $\mathcal{Y}$ of length $m+1$, there must be two runs ψ_j and ψ_i with $i \neq j$ such that $\psi_i(m) = \psi_j(m)$. This immediately implies, by definition of Y-run, that the ω-word

$$\alpha' = \alpha_j(0, 2^{m+1} - 2) \cdot \alpha_i(2^{m+1} - 1, \omega)$$

has a succesful Y-run, namely ψ such that $\psi(k) = \psi_j(k)$, for $0 \leq k \leq m$ and $\psi = \psi_i$, elsewhere. This lead to a contradiction since $\alpha' \notin L$. □

Proposition 5 *The emptiness problem for Y-tree ω-languages is not decidable.*

Proof. In order to show that emptiness is undecidable, it is sufficient to realise that for a given Y-tree automaton $\mathcal{Y}$ (for finitary words on the alphabet Σ) we can construct a Y-tree automaton $\mathcal{Y}'$ for ω-words on alphabet $\Sigma \cup \{\sharp\}$ (with $\sharp$ not in Σ), such that the language accepted by $\mathcal{Y}$ is empty iff the language accepted by $\mathcal{Y}'$ is empty. Now, the undecidability of emptiness problem for the case of ω-words follows from the undecidability of the emptiness problem in the finitary case (see [6]). □

To define a calculus which is as powerful to accept Y-tree ω-languages, we extend $MSO[<]$ by a (partial) function $adj : \mathbb{N} \mapsto \mathbb{N}$, s.t.

$$\text{if } x = 2^{k_n} + 2^{k_{n-1}} + \ldots + 2^{k_0}, \text{ with } k_n > k_{n-1} > \ldots > k_0 > 0, \text{ then}$$
$$adj(x) = x + 2^{k_0} + 2^{k_0 - 1}.$$

The logical definability of ω-languages accepted by Y-tree automata can be proved by showing that the function adj allows us to impose on natural numbers the upward unbound Y-tree structure depicted in figure 11. Such a structure is obtained by enriching the binary tree structure of figure 5 with an edge between x and y iff $adj(x) = y$.
In fact, $MSO[<, adj]$ allows one to define two predicates $Right\_son(z, x)$ and $Left\_son(z, x)$, which are true iff the number x is the right and left son, respectively, of z (in the sense of Eq.2 and Eq.1).
Now, x is the left son of z, if either $x + 1 = z$ and z is at the 1-th layer (i.e. $\exists s(adj(z) = s \wedge \neg\exists t(adj(s) = t))$), or z is at the i-th layer (with $i > 1$) and

$$x = min\{s : s < z \wedge adj(s) > z)\}.$$

The two cases above can be expressed by a formula of $MSO[<, adj]$ which, for simplicity, we denote by $Left\_son(z, x)$. The formula $Right\_son(z, x)$ can be expressed in terms of adj and $Left\_son(z, x)$ as follows

$$\neg\exists s(adj(x) = s) \wedge x = z + 1 \vee$$
$$\exists s, t(adj(x) = s \wedge adj(z) = t \wedge Left\_son(t, s)).$$

Example 3 *A $EMSO[<, adj]$-formula defining the language L of Eq.7 is*

$$\exists Y(\forall x(Y(x) \Rightarrow Q_1(x)) \wedge$$
$$\forall y, z(son(z, y) \wedge Y(y) \Rightarrow \forall s(son(z, s) \Rightarrow Y(s)))),$$

where $son(z, y)$ stands for $Left\_son(z, y) \vee Right\_son(z, y) \vee adj(z) = y$.

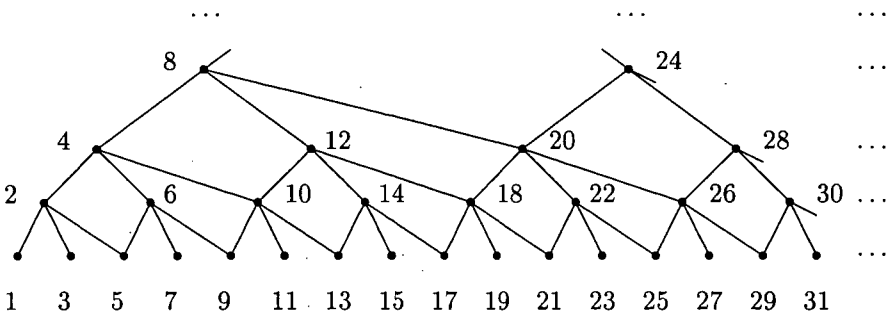


Figure 11: The upward unbounded Y-tree structure on $\mathbb{N}$.

Now, we can state the main theorem of this section, namely the logical definability of Y-tree ω-languages. As it is precisely shown in the proof of Th. 6, the upward unbounded Y-tree structure on $\mathbb{N}$ allows to simulate a Y-tree systolic computation. Also in this case, as remarked in the end of section 2, an ω-word α cannot be associated directly with the 0-th layer of the structure but it is associated to this layer only after a preprocessing step.

Theorem 6 *Any ω-language accepted by a Y-tree automaton is $EMSO[<, adj]$-definable.*

Proof. Assume without loss of generality that the set of states of the Y-tree automaton $\mathcal{Y}$ is $Q = 1, \ldots, m$. A Y-tree computation can be obtained by labelling, compatibly with the transition relation f, the upward unbounded Y-tree structure with elements of Q. Such a labelling is defined by partitioning the set of natural numbers into m sets $Y_1, \ldots, Y_m$ and assuming that a number x belongs to Y_i iff the state i labels the node x of the upward unbounded Y-tree structure. We introduce some short notations:
$Odd(x)$ (i.e. x is an odd number) stands for $\neg\exists z(adj(x) = z)$.
$Power(x)$ (i.e. x is a power of two) stands for
$x = 1 \vee \exists z(adj(x) = z \wedge \forall y, s((y < x \wedge adj(y) = s) \Rightarrow s < z))$.
$x \xrightarrow{L} i$ (i.e. the left son of x is associated with the state i) stands for
$\exists y\,(Left\_son(x, y) \wedge Y_i(y))$.
$x \xrightarrow{R} i$ (i.e. the right son of x is associated with the state i) stands for
$\exists y\,(Right\_son(x, y) \wedge Y_i(y))$.
$x \xrightarrow{A} i$ (i.e. the adoptive son of x is associated with the state i) stands for
$\exists y\,(adj(x) = y \wedge Y_i(y))$
Given a Y-tree automaton $\mathcal{Y} = \langle\{1, \ldots, m\}, Q_0, g, f\rangle$, a formula ϕ belonging to $EMSO[<, adj]$ accepting $\mathcal{L}_\omega(\mathcal{Y})$ is the following:
$\exists Y_1, \ldots, Y_m$ (

$$\bigwedge_{i \neq j} \neg\exists y\,(Y_i(y) \wedge Y_j(y)) \wedge \tag{10}$$

$$\forall x \left(Odd(x) \Rightarrow \left(\bigwedge_{a,b,c \in \Sigma} Q_a(x-1) \wedge Q_b(x) \wedge Q_c(x+1) \right) \Rightarrow \bigvee_{\{i:\langle j,k,l,i\rangle \in f, \langle a,j\rangle, \langle b,k\rangle, \langle c,l\rangle \in g\}} Y_i(x) \right) \right) \wedge \tag{11}$$

$$\forall z \left(\bigwedge_{i,j,l \in Q} \left((z \xrightarrow{L} i \wedge z \xrightarrow{R} j \wedge z \xrightarrow{A} l) \Rightarrow \bigvee_{\{k:\langle i,j,l,k\rangle \in f\}} Y_k(z) \right) \right) \wedge \tag{12}$$

$$\bigvee_{i \in F} \forall x \exists y \, (x < y \wedge Power(y) \wedge Y_i(y)) \tag{13}$$

). □

As a consequence of Theorem 6 and the undecidability of emptiness for Y-tree ω-languages (see Proposition 5), we have the following corollary.

Corollary 7 *The theory $EMSO[<, adj]$ is undecidable.*

The theory $EMSO[<, adj]$ extends $MSO[<, flip]$ since the class of Y-tree ω-languages strictly includes the class of binary tree ω-languages (see Theorem 2) and $MSO[<, flip]$ is a characterisation of that class of languages (see Theorem 1).

Corollary 8 *$EMSO[<, adj]$ is a proper extension of $MSO[<, flip]$.*

The function *adj* we have chosen to extend $MSO[<]$ seems not to be more expressive than it is required to define Y-tree ω-languages. This is suggested by the fact that the ω-language encoding the function *adj* (as well as its complement) can be accepted by a Y-tree automaton. Given a relation $R \subseteq \mathbb{N} \times \mathbb{N}$, the ω-word on the alphabet $\{0,1,2\}$ *encoding the pair* $\langle i,j \rangle \in R$ is $\alpha_{\langle i,j \rangle} \in \{0,1,2\}^\omega$ such that $\alpha_{\langle i,j \rangle}(i) = 1$, $\alpha_{\langle i,j \rangle}(j) = 2$ and $\alpha_{\langle i,j \rangle}(k) = 0$ for all $k \neq i, j$. The ω-language *encoding the relation* R is the set $\{\alpha_{\langle i,j \rangle} : \langle i,j \rangle \in R\}$.

Proposition 9 *There exists a deterministic Y-tree automaton accepting the ω-language L encoding the adj function.*

Proof. We define two Y-tree automata $\mathcal{Y}_1$ and $\mathcal{Y}_2$ such that $L = \mathcal{L}_\omega(\mathcal{Y}_1) \cup \mathcal{L}_\omega(\mathcal{Y}_2)$. The automaton $\mathcal{Y}_1$ (resp. $\mathcal{Y}_2$) accepts ω-words having the form

$$0^{2^{k_n}+\ldots+2^{k_0}}.1.0^{2^{k_0}+2^{k_0-1}-1}.2.0^\omega$$

with $k_n > k_{n-1} > \ldots > k_0$ and $k_0 = 1$ (resp. $k_0 > 1$). The claim follows from closure under union of languages accepted by Y-trees. As concerns the automaton $\mathcal{Y}_1$, notice that the node of the Y-tree unbound structure that receives the input 1 (resp. 2) must be the left son of a right son node (resp. the right son of a left son node). Such a pattern can be detected in two steps by the following automaton:

$$\mathcal{Y}_1 = \langle \{0,1,2\}, \{0,1,2,\overline{1},\overline{2},s\}, \{s\}, g_1, f_1 \rangle, \text{ where}$$

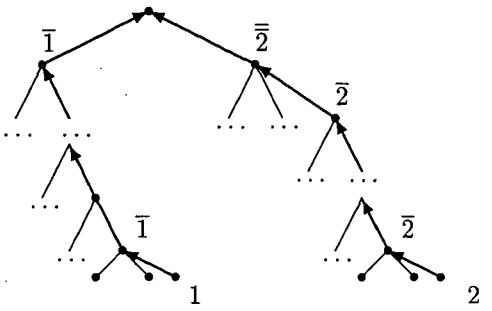


Figure 12: The pattern recognised by automaton $\mathcal{Y}_2$ of Proposition 9.

- g_1 is the identity;
- $f_1 = \{ \langle 0,0,0,0\rangle, \langle 0,0,1,\overline{1}\rangle, \langle 0,2,0,\overline{2}\rangle, \langle 1,0,0,0\rangle, \langle \overline{1},0,\overline{2},s\rangle,$
  $\langle \overline{2},0,0,0\rangle, \langle 0,0,\overline{1},0\rangle, \langle s,0,0,s\rangle, \langle 0,s,0,s\rangle, \langle 0,0,s,0\rangle\};$

The pattern detected by automaton $\mathcal{Y}_2$ is depicted in figure 12. The symbol 1, received at input level, must be propagated the first time along an adoptive edge and then along a right edge as far as possible. The symbol 2 must be propagated the first time along an adoptive edge, then along a right edge as far as possible, and, finally, again along an adoptive edge. Signals propagated in this way must be eventually collected by a node from the left and right son. The automaton $\mathcal{Y}_2$ is as follows:

$$\mathcal{Y}_2 = \langle \{0,1,2\}, \{0,1,2,\overline{1},\overline{2},\overline{\overline{2}},s\}, \{s\}, g_2, f_2\rangle, \text{ where}$$

- g_2 is the identity;
- $f_2 = \{ \langle 0,0,1,\overline{1}\rangle, \langle 0,\overline{1},0,\overline{1}\rangle, \langle 1,0,0,0\rangle, \langle 0,0,\overline{1},0\rangle,$
  $\langle 0,0,2,\overline{2}\rangle, \langle 0,\overline{2},0,\overline{2}\rangle, \langle 0,0,\overline{2},\overline{\overline{2}}\rangle, \langle 2,0,0,0\rangle, \langle \overline{2},0,0,0\rangle,$
  $\langle \overline{1},\overline{\overline{2}},0,s\rangle, \langle s,0,0,s\rangle, \langle 0,s,0,s\rangle, \langle 0,0,s,0\rangle\}.$

In general, if $\alpha(i) = 1$, then $\overline{1}$ flows up to k-th layer and labels a left-son processor iff $i = 2^k + m2^{k+1}$ with $(m \geq 0)$. If $\alpha(j) = 2$, the signal flows up to the k-th layer labelling a right-son processor iff $j = 2^{k-1} + m2k + 1$ (with $m \geq 1$). If i and j are the left and right son of the same node, then $i = 2^k + m2^{k+1}$ and $j = 2^{k-1} + (m+1)2^{k+1}$ and then $j = i + 2^{k-1} + 2^k = adj(i)$. □

Since Y-tree ω-languages are closed under union and intersection, by Proposition 3, any formula freely constructed from atomic formulas using the boolean connectives $\wedge$, $\vee$ and $\neg$ has a corresponding Y-tree automaton. If ϕ is a formula with a corresponding Y-tree automaton, also the existential quantification of ϕ (on both individual and predicates) has a Y-tree automaton as a counterpart since Y-tree ω-languages are closed under projection. The problem whether any formula $\neg\exists x\phi$,

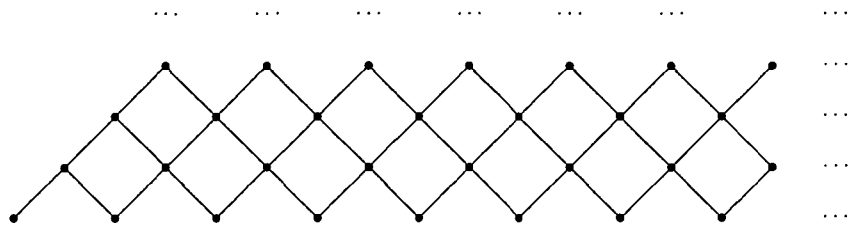


Figure 13: An upward unbounded trellis structure.

with x an individual variable and ϕ a first order formula, can have a corresponding Y-tree automaton, remains an open problem. If such a problem had positive answer, we would have that $EMSO[<, adj]$ would be a characterisation of Y-tree ω-languages, namely that an ω-language is accepted by a Y-tree automaton iff it is $EMSO[<, adj]$-definable.

5 Trellis ω-Languages

The expressive power and properties of trellis automata as acceptors of finitary languages (over the standard interconnection structure of figure 1.c) have been studied in [1, 2] for the deterministic case and in [9] for the non-deterministic case. In this section we introduce and study non-deterministic trellis automata as acceptors of ω-languages. In this respect, a trellis automaton processes a word exactly as a binary tree (or a Y-tree) automaton does with the only exception that it uses, for the computation, an upward unbounded trellis structure (see figure 13).

Definition 4 *A* trellis automaton *is a tuple* $\mathcal{T} = \langle \Sigma, Q, Q_0, g, f \rangle$, *where*

- Q *is the finite set of* states *and* $Q_0 \subseteq Q$ *is the set of* final states;
- $g \subseteq \Sigma \times Q$ *is the* input relation;
- $f \subseteq Q \times Q \times Q$ *is the* transition relation.

A computation of a trellis automaton is then a labelling of an upward unbounded trellis structure compatible with the input and transition relations. As in the case of binary tree and Y-tree automata, we define a step-wise version of such a computation. Given an ω-word α, at each computation step, the automaton processes a symbol of α. So, at the i-th step (with $i \geq 0$) the prefix $\alpha(0, i)$ of α is processed and the result of such a computation is stored into a sequence $q_0 \ldots . q_i$ of states of Q. The sequence $q_0 \ldots . q_i$ is precisely the sequence of states labelling the path of right edges, from the i-th element of the 0-th column of the upward unbounded trellis structure, leading to the 0-th layer. Now, the $i+1$-th step

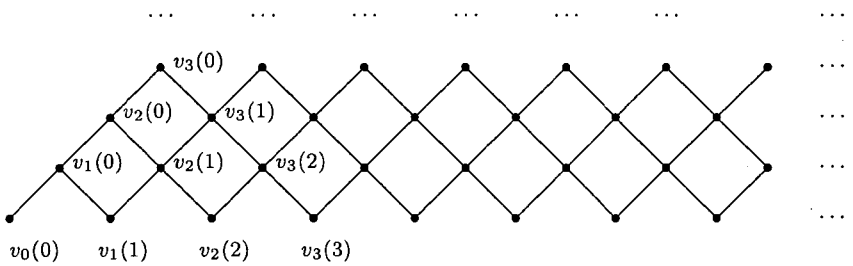


Figure 14: The structure of a T-run.

transforms the sequence $q_0 \ldots . q_i$, encoding the computation at the i-th step, into a sequence of states $q'_0 \ldots . q'_i, q'_{i+1}$ as a result of composing $q_0 \ldots . q_i$ with the result of processing the symbol $\alpha(i+1)$. For a trellis automaton $\mathcal{T} = \langle \Sigma, Q, Q_0, g, f \rangle$, the transformation above, called *run step*, is defined by the relation

$$O_{\mathcal{T}} \subseteq Q^\star \times \Sigma \times Q^\star,$$

which is such that $\langle v, a, u \rangle \in O_{\mathcal{T}}$ implies $|v| = k$ and $|u| = k+1$, for some $k \geq 0$ and is defined recursively on k as follows:

- if $k = 0$, then $\langle \epsilon, a, q \rangle \in O_{\mathcal{T}}$, with $\langle a, q \rangle \in g$;
- if $k > 0$, then $\langle v, a, u \rangle \in O_{\mathcal{T}}$, where
  - $\langle a, u(k) \rangle \in g$ and $\langle v(i), u(i+1), u(i) \rangle \in f$, for $0 \leq i < k$.

A *T-run* for an ω-word α stores the result of each computation step and it is defined as a map $\psi : \mathbb{N} \mapsto Q^\star$ such that:

$$\langle \alpha(0), \psi(0) \rangle \in g \text{ and } \langle \psi(i), \alpha(i+1), \psi(i+1) \rangle \in O_{\mathcal{T}}, \text{ for } i \geq 0.$$

A T-run ψ is *successful* iff there are infinitely many $i \in \mathbb{N}$ such that $\psi(i) = v_i$ and $v_i(0) \in Q_0$.
The structure of a T-run is depicted in figure 14.

Example 4 *An instance of trellis ω-language is:*

$$L = \{\alpha \in \{0,1\}^\omega : \alpha(i)\alpha(i+k) \textit{ for some } k \geq 1, \textit{ for all } i \geq 0\}. \qquad (14)$$

Notice that $\alpha \in L$ iff $\alpha = w^\omega$ with $w \in \{0,1\}^\star$. Before giving a formal definition of the automaton accepting the language L above, we sketch the idea of the recognising process. Assume that $\alpha = w^\omega$, with $|w| = k$. Each input symbol is propagated upward along the left and the right edges till it reaches the $k-1$-th layer. Each processor of the k-th layer checks whether the symbols, received from the left and the right sons, agree. Processors in the next layer check that all their sons had a

successful result and so on. A processor chooses non-deterministically whether to propagate the signals it is receiving or to perform the test, and the computation can be successful only if all of the processors of the k-th layer non-deterministically choose to perform the test.

The automaton $\mathcal{T} = \langle \Sigma, Q, Q_0, g, f \rangle$ *accepting* L *has the following components:*

- $Q = \{ok, x, [x,y] : x, y \in \{0,1\}\}$, $Q_0 = \{ok\}$, $g = \{\langle x, x\rangle : x \in \{0,1\}\}$;
- $f = \{\langle x, y, [y,x]\rangle, \langle [x,y], [z,t], [z,y]\rangle, \langle [x,y], [y,z], ok\rangle, \langle ok, ok, ok\rangle : x, y, z, t \in \{0,1\}\}$.

States $[x,y]$ *are used to forward properly the input symbols upward along the left and right edge, respectively. Each node of the* 1*-th layer enters the state* $[y,x]$ *after receiving symbol* x *from the left-son and symbol* y *from the right-son. Each processor, after receiving state* $[x,y]$ *from the left-son and state* $[z,t]$ *from the right-son, propagates the symbols* y *and* z *by entering the state* $[z,y]$. *An example of successful computation is shown in figure 15 for* $k = 3$.

We consider now the expressive power of trellis automata with respect to Y-tree automata.

Theorem 10 *The class of trellis* ω*-languages strictly includes the class of* Y*-tree* ω*-languages.*

Proof. We start showing that a systolic computation over an upward unbounded Y-tree structure can be simulated over an upward unbounded trellis structure.

Let $\mathcal{Y}$ be a Y-tree automaton, we outline the construction of a trellis automaton $\mathcal{T}$ accepting the same ω-language of $\mathcal{Y}$. The behaviour of the node of $\mathcal{Y}$ placed in the i-th layer and j-th column of the upward unbounded Y-tree structure is simulated by the behaviour of the node of the automaton $\mathcal{T}$ placed in the $j(3^i - 3^{i-1})$-th column and the $3^i - 3^{i-1}$-th layer of the upward unbounded trellis structure (for $i \geq 1$, $j \geq 0$). With reference to figure 16, simulating nodes are labelled by symbol x. The left, right and adoptive edges connecting a node with its children in the upward unbounded Y-tree structure are simulated by the left, right and adoptive paths (in bold in the figure). Notice that the states coming from a left and right x-labelled node are collected by a y-labelled node and then forwarded to the x-labelled father node. Let us consider now the complete labelling of the figure 16 given in figure 17. States t_0 an t_1 label nodes in the left and right path, respectively, from a x-labelled node to a y-labelled node. State t_1 labels nodes in the path between a y-labelled and a x-labelled state. State t_2 labels nodes in the adoptive path between two x-labelled nodes. The nodes in the region bounded by a t_1-labelled and t_2-labelled path (i.e. the region where a left and right son are collected) are labelled by symbol b. The region of nodes where an adoptive son is transmitted (bounded by a t_2-labelled path) are labelled by symbol a. All the remaining nodes are labelled by symbol c. Now, it is not difficult to define an automaton whose successful computations are exactly those producing the above described labelling

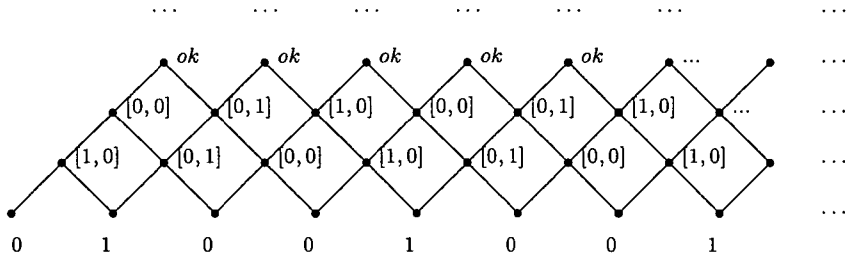


Figure 15: A successful computation for the ω-word $(010)^\omega$.

of the upward unbounded trellis structure. Such an automaton can then be easily adapted to simulate any given Y-tree automaton.

We prove now that the inclusion is proper, by showing that the trellis ω-language L of Example 4 is not a Y-tree ω-language. The proof is by contradiction. Let us assume that there exists a Y-tree automaton $\mathcal{Y}$ having n states and accepting L. For a suitable m, we take now $s = 2^{2^{m+1}-1} > n^{m+1}$ (different) ω-words $\alpha_1, \ldots, \alpha_s$ of L such that $\alpha_i = w_i{}^\omega$, $|w_i| = 2^{m+1} - 1$, $w_i \neq w_j$, $1 \leq i \neq j \leq s$. Let ψ_i be a successful Y-run of $\mathcal{Y}$ for α_i (with $1 \leq i \leq s$). Since there are at most $n^{m+1} < s$ different strings of states of length $m+1$, there must be two runs ψ_j and ψ_i with $i \neq j$ such that $\psi_i(m) = \psi_j(m)$. This immediately implies, by definition of Y-run, that the ω-word

$$\alpha' = \alpha_j(0, 2^{m+1} - 2) \cdot \alpha_i(2^{m+1} - 1, \omega) = w_j \cdot w_i{}^\omega$$

has a successful Y-run, namely the run ψ such that $\psi(k) = \psi_j(k)$, for $0 \leq k \leq m$ and $\psi = \psi_j$, elsewhere. This leads to a contradiction since $\alpha' \notin L$. □

Since the emptiness problem is undecidable for the class of Y-tree ω-languages we have the following corollary.

Corollary 11 *The emptiness problem for the class of trellis ω-languages is not decidable.*

By applying standard techniques, the following closure properties can be proved.

Proposition 12 *The class of trellis ω-languages is closed under union, intersection and projection.*

Moreover, it remains an open question whether the class of trellis ω-languages is closed under complementation. Also in the case of finitary languages accepted by trellis automata the closure under complementation remains still an hard open question. In [9] it is suggested that the question has probably negative answer since, otherwise, a positive answer would imply the closure under complementation of the well known complexity class NP.

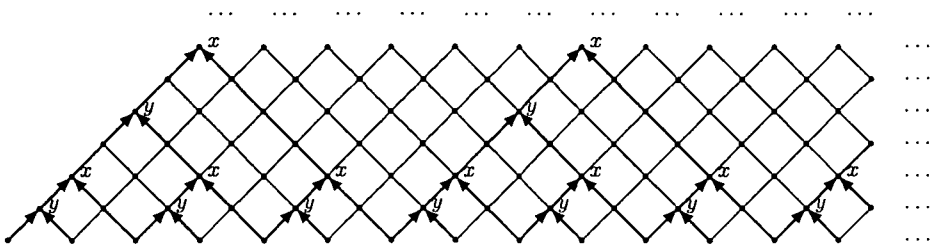


Figure 16: Embedding of the upward unbounded Y-tree structure in the upward unbounded trellis structure.

Let us consider now the logical definability of trellis ω-languages. In order to capture the class of ω-languages accepted by trellis automata we extend $MSO[<]$ with the function $2x$ giving, for a number x, its double.

Example 5 *We show that the language encoding the function adj can be defined in* $MSO[<, 2x]$. *With reference to the upward unbound* Y*-tree structure, it is easy to see that if* x *belongs to the column* c *which includes the odd number* k, *then* $adj(x)$ *belongs to the column* c' *which includes the odd number* $2k+3$. *Notice also that if* c *is the* i*-th column (with* $i > 0$*), then* $adj(x)$ *is the least element in* c' *greater than* x. *Otherwise, if* c *is the* 0*-th column, then* $adj(x)$ *is the double of the least element in* c' *greater than* x.
The formula $Column(x, y)$ *(i.e.* y *belongs to the the column containing the odd number* x*) stands for*

$$Odd(x) \wedge \exists X(X(x) \wedge X(y) \wedge$$
$$\forall z, s(X(z) \Rightarrow X(2z) \wedge Odd(z) \Rightarrow z = x \wedge (X(s) \wedge 2z = s) \Rightarrow X(z)),$$

where $Odd(x)$ *stands for* $\neg\exists y(x = 2y)$.
Now, the formula defining the function adj is as follows:

$$\begin{array}{c}\exists x, y(x < y \wedge Q_1(x) \wedge Q_2(y) \wedge \forall z(z \neq x \wedge z \neq y \Rightarrow Q_0(z)) \wedge \\ \exists z, s(Column(z, x) \wedge Column(s, y) \wedge s = 2z + 3 \wedge \\ (z = 1 \Rightarrow \exists h(Column(s, h) \wedge x < h \wedge y = 2h \wedge \\ \forall r(Column(s, r) \wedge x < r \Rightarrow h \leq r))) \wedge \\ (z > 1 \Rightarrow \forall h(Column(s, h) \wedge x < h \Rightarrow h \leq y))),\end{array}$$

with $z = 1$ *and* $z > 1$ *obvious shorthands.*

As an immediate consequence of Example 5 we have the following expressiveness property.

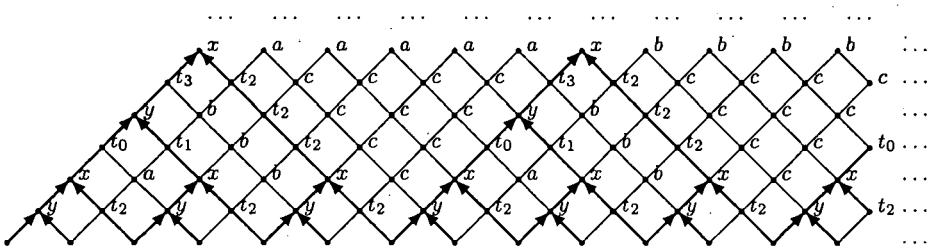


Figure 17: A Y-tree labelling of an upward unbounded trellis structure

Proposition 13 *$MSO[<, 2x]$ is at least as expressive as $EMSO[<, adj]$.*

Notice that if $EMSO[<, adj]$ were a characterization of Y-tree ω-languages, then $MSO[<, 2x]$ would be a proper extension of $EMSO[<, adj]$. The double function allows impose on the set of natural numbers an upward unbounded trellis structure (see figure 18). The sets identifying the i-th layer and column of the structure are defined exactly as in Eq.5 and Eq.6, respectively. The left son of a number x is the half part of x. As concerns the right son, we proceed as follows. If x belongs to the 0-th column, then the right son of x is

$$y = min\{z : z \text{ belongs to the 1-th column}, z > x\}.$$

If x belongs to the i-th column (with $i > 0$), then the right son is

$$y = max\{z : z \text{ belongs to the } i + 1 - \text{th column}, z < x\}.$$

Theorem 14 *Any ω-language accepted by a trellis automaton is $MSO[<, 2x]$-definable.*

Proof. Assume without loss of generality that the set of states of $\mathcal{T}$ is $Q = 1, \ldots, m$. The idea of the proof is similar to that of Theorem 6. In particular, we shall consider predicates $Y_1, \ldots, Y_m$ assuming that a number x belongs to Y_i iff the state i labels the node x of the unbounded trellis structure. We introduce some short notations:
$x \xrightarrow{L} i$ (i.e. the left son of x is associated with the state i) stands for
$\exists y(2y = x \wedge Y_i(y))$.
$Right - son(x, y)$ (i.e. y is the right son of x) stands for

$$\exists r(Column(r, x) \wedge Column(r + 2, y) \wedge$$
$$(r = 1 \Rightarrow y > x \wedge \forall s(Column(v + 2, s) \wedge s > x \Rightarrow y \leq s)) \wedge$$
$$(r > 1 \Rightarrow y < x \wedge \forall s(Column(r + 2, s) \wedge s < x \Rightarrow y \geq s))).$$

$x \xrightarrow{R} i$ (i.e. the right son of x is associated with the state i) stands for
$\exists y\,(Right - son(x,y) \wedge Y_i(y))$.
Given a trellis automaton $\mathcal{T}$, a formula $\phi \in MSO[<, 2x]$ accepting $\mathcal{L}_\omega(\mathcal{T})$ is the following:
$\exists Y_1, \ldots, Y_m$ (

$$\bigwedge_{i \neq j} \neg \exists y\,(Y_i(y) \wedge Y_j(y)) \wedge \tag{15}$$

$$\forall x \left(Odd(x) \Rightarrow \left(\bigwedge_{a,b \in \Sigma} (Q_a(x-1) \wedge Q_b(x)) \Rightarrow \bigvee_{\{i : \langle j,k,i \rangle \in f, \langle a,j \rangle, \langle b,k \rangle \in g\}} Y_i(x) \right) \right) \wedge \tag{16}$$

$$\forall z \left(\bigwedge_{i,j \in Q} \left((z \xrightarrow{L} i \wedge z \xrightarrow{R} j) \Rightarrow \bigvee_{\{k : \langle i,j,k \rangle \in f\}} Y_k(z) \right) \right) \wedge \tag{17}$$

$$\bigvee_{i \in F} \forall x \exists y\,(x < y \wedge Column(1,y) \wedge Y_i(y)) \tag{18}$$

). □

The fact that $MSO[<, 2x]$ is undecidable, is well known (see [4, 14]). Moreover, if trellis ω-languages were closed under complementation, then $MSO[<, 2x]$ would be a characterisation of trellis ω-languages. This would follow from the fact that the ω-language which encodes the function *double* (as well as its complement) can be accepted by a trellis automaton, and from the closure of trellis ω-languages under union, intersection and projection (see Proposition 12).

Proposition 15 *There exists a deterministic trellis automaton accepting the ω-language encoding the double function.*

Proof. We define a deterministic trellis automaton $\mathcal{T} = \langle \Sigma, Q, Q_0, f, g \rangle$ which accepts ω-words having the form $0^x 1.0^{x-1}.2.0^\omega$ with $x \geq 1$. This automaton is as follows:

- $\Sigma = \{0, 1, 2\}$;
- $Q = \{0, 1, m_l, m_r, ok\}$; $Q_0 = \{ok\}$;
- $g = \{\langle 0,0 \rangle, \langle 1, m_r \rangle, \langle 2,1 \rangle\}$;
- $f = \{\ \langle 0,0,0 \rangle, \langle 0, m_r, m_l \rangle, \langle m_r, 0, 0 \rangle, \langle 0, m_l, m_0 \rangle, \langle m_l, 0, m_r \rangle,$
  $\langle 0,1,1 \rangle, \langle 1,0,0 \rangle, \langle m_r, 1, 1 \rangle, \langle m_l, 1, ok \rangle, \langle ok, 0, ok \rangle\}$.

□

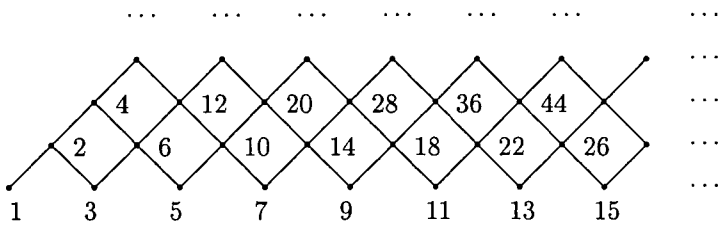


Figure 18: The upward unbounded trellis structure on $\mathbb{N}$.

6 Conclusions

In this paper we have introduced the classes of ω-languages accepted by Y-tree and trellis systolic automata, and we have studied some of their expressiveness, closure and decidability properties. We have, then, defined two monadic second order logics, $EMSO[<, adj]$ and $MSO[<, 2x]$, which allow to logically define the two mentioned classes of languages. The problem whether $EMSO[<, adj]$ is a characterisation of the class of ω-languages accepted by Y-tree automata remains open, and the first step of our future work will be devoted to solve it. The question whether $MSO[<, 2x]$ is a characterisation of the class of ω-languages accepted by trellis languages is, conversely, a problem which hardly will have an answer. Moreover, this paper provides the theoretical framework for investigating expressiveness issues of monadic second theories of time granularity interpreted on different types of ω-layered metric temporal structures (see [11] for a discussion on the open related questions). Our future work will consider also such an investigation.

References

[1] K. Culik II, J. Gruska, A. Salomaa, Systolic trellis automata. Part I, *International Journal of Computer Mathematics*, **15** (1984) 195–212.

[2] K. Culik II, J. Gruska, A. Salomaa, Systolic trellis automata, International *Journal of Computer Mathematics*, **16** (1984) 3–22.

[3] K. Culik II, J. Gruska, A. Salomaa, Systolic trellis automata: Stability, decidability and complexity, *Information and Control*, **71** (1986) 218–230.

[4] C.C. Elgot, M.O. Rabin, Decidability and undecidability of extensions of secon (first) order theory of (generalized) successor, *The Journal of Symbolic Logic*, **31** (1966) 169–181.

[5] E. Fachini, M. Napoli, C-tree systolic automata, *Theoretical Computer Science*, **56** (1988) 155–186.

[6] E. Fachini, R. Franes, M. Napoli, M. Parente, BC-tree systolic automata: characterization and properties, *Journal of Computer and Artificial Intelligence*, **1** (1989) 53–82.

[7] E. Fachini, A. Monti, Chomsky hierarchy and systolic Y-tree automata, *Funtamenta Informaticae*, **29** (1996) 325–339.

[8] J. Gruska, Synthesis, structure and power of systolic computations, *Theoretical Computer Science*, **71** (1990) 47–78.

[9] H. Ibarra, S.M. Kim, Characterizations and computational complexity of systolictrellis automata, *Theoretical Computer Science*, **29** (1984) 123–153.

[10] A. Montanari, A. Peron, A. Policriti, Theories of ω-layered metric temporal structures: Expressiveness and decidability, *Logic Journal of the IGPL*, **7** (1999) 79–102.

[11] A. Montanari, A. Peron, A. Policriti, The way to go: Multi-level temporal logics, Proceedings of IWTS'99: 1st International Workshop on Specification and Verification of Timed Systems, N. Yonezaki (Ed.) Kyoto Research Institute of Mathematical Science, march 1999.

[12] A. Monti, A. Peron, A logical characterization of systolic languages, in: *Proc. STACS'98*, LNCS Vol. 1373 (Springer, Berlin,1998), 466–476.

[13] A. Monti, A. Peron, Systolic tree ω-languages: The operational and the logical view, *Theoretical Computer Science*, **233** (2000) 1–18.

[14] R.M. Robinson, Restricted set-theoretical definitions in aritmetic, Proc. Amer. Mat. Soc. Vol. 9, 1958, 238–242.

[15] M.Y. Vardi, Nontraditional applications of automata theory, LNCS Vol. 789 (Springer, Berlin,1994), 575–597.

[16] W. Thomas, Automata on infinite objects, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B*, (Elsevier, Amsterdam, 1990) 133–191.

[17] W. Thomas, Languages, Automata and Logic, in: G. Rozemberg and A. Salomaa, eds., *Handbook of formal languages, Vol. 3*, Springer, 1997, 389–455.

Received March, 2001

Acta Cybernetica **15** (2001) 101-106.

Affine matching of two sets of points in arbitrary dimensions

Attila Tanács * Gábor Czédli † Kálmán Palágyi ‡
Attila Kuba §

Abstract

In many applications of computer vision, image processing, and remotely sensed data processing, an appropriate matching of two sets of points is required. Our approach assumes one–to–one correspondence between these sets and finds the optimal global affine transformation that matches them. The suggested method can be used in arbitrary dimensions. A sufficient existence condition for a unique transformation is given and proven.

1 Introduction

Many applications lead to the following mathematical problem: Two corresponding sets of points $\{p_i\}$ and $\{q_i\}$ $(i = 1, 2, \ldots, n)$ are given in the k–dimensional Euclidean space $\mathbb{R}^k$, and the transformation $T : \mathbb{R}^k \to \mathbb{R}^k$ is to be found that gives the minimal mean squared error

$$\sum_{i=1}^{n} \|T(q_i) - p_i\|^2.$$

The dimension k is usually 2 or 3. Some solutions have been proposed for this problem assuming rigid–body transformation (i.e., where only rotations and translations are allowed) [1, 3, 6, 7, 13], affine transformation (i.e., which maps straight lines to straight lines, parallelism is preserved, but angles can be altered) [8], and non-linear transformation (i.e., which can map straight lines to curves) [2, 5, 8]. In [10], a solution is proposed when the correspondence between the

*Department of Applied Informatics, University of Szeged, H–6701 Szeged P.O.Box 652, Hungary, e–mail: tanacs@inf.u-szeged.hu

†Bolyai Institute, University of Szeged, H–6720 Szeged, Aradi vértanúk tere 1, Hungary, e–mail:czedli@math.u-szeged.hu

‡Department of Applied Informatics, University of Szeged, H–6701 Szeged P.O.Box 652, Hungary, e–mail: palagyi@inf.u-szeged.hu

§Department of Applied Informatics, University of Szeged, H–6701 Szeged P.O.Box 652, Hungary, e–mail: kuba@inf.u-szeged.hu

point sets is unknown, assuming affine transformation. It is mentioned, that if the correspondence was known, a simpler solution is possible e.g., using least squares method, but neither such a method nor a sufficient existence condition for unique solution is given or referenced.

In this paper, we present a method for solving the problem assuming affine transformation, which can be used in arbitrary dimensions. The method is described in Section 2. We state and prove a sufficient existence condition for a unique solution in Section 3. A related open problem concerning degeneracy is presented in Section 4.

2 Method for affine matching of two sets of points

Given a matrix

$$\mathcal{T} = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1k} & t_{1,k+1} \\ t_{21} & t_{22} & \cdots & t_{2k} & t_{2,k+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{k1} & t_{k2} & \cdots & t_{kk} & t_{k,k+1} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix},$$

it determines an affine transformation $T : \mathbb{R}^k \to \mathbb{R}^k$ as follows: For $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$ in $\mathbb{R}^k$ we have $y = T(x)$ if and only if

$$\begin{pmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{ik} \\ 1 \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1k} & t_{1,k+1} \\ t_{21} & t_{22} & \cdots & t_{2k} & t_{2,k+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{k1} & t_{k2} & \cdots & t_{kk} & t_{k,k+1} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ik} \\ 1 \end{pmatrix}.$$

Note that homogeneous coordinates are used. Each affine transformation T can uniquely be represented in this form [4]. The transformation has $k \cdot (k+1)$ degrees of freedom according to the non–constant matrix elements.

Let us fix an affine transformation $T : \mathbb{R}^k \to \mathbb{R}^k$ and the corresponding $\mathcal{T}$ as above. Let $\{p_i\}$ and $\{q_i\}$ be two sets of n points, where

$$\begin{aligned} p_i &= (p_{i1}, p_{i2}, \ldots, p_{ik}) \in \mathbb{R}^k \quad \text{and} \\ q_i &= (q_{i1}, q_{i2}, \ldots, q_{ik}) \in \mathbb{R}^k \quad (i = 1, 2, \ldots, n). \end{aligned}$$

Let $\{p'_i\}$ be a set of n points in $\mathbb{R}^k$, where $p'_i = T(q_i)$ $(i = 1, 2, \ldots, n)$. Define the merit function $\mathcal{S}$ of $k \cdot (k+1)$ variables as follows:

$$\mathcal{S}(t_{11}, \ldots, t_{k,k+1}) = \sum_{i=1}^{n} \|p'_i - p_i\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{k} (t_{j1} \cdot q_{i1} + \ldots + t_{jk} \cdot q_{ik} + t_{j,k+1} - p_{ij})^2.$$

which is generally regarded as the *matching error*.

The least square solution of matrix $\mathcal{T}$ is determined by minimizing the function $\mathcal{S}$. Function $\mathcal{S}$ may be minimal if all of the partial derivatives $\frac{\partial \mathcal{S}}{\partial t_{11}}, \ldots, \frac{\partial \mathcal{S}}{\partial t_{k,k+1}}$ are equal to zero. The required $k \cdot (k+1)$ equations:

$$\frac{\partial \mathcal{S}}{\partial t_{uv}} = 2 \cdot \sum_{i=1}^{n} q_{iv} \cdot (t_{u,k+1} - p_{iu} + \sum_{l=1}^{k} t_{ul} \cdot q_{il}) = 0$$
$$(u = 1, 2, \ldots, k\,,\ v = 1, 2, \ldots, k),$$

$$\frac{\partial \mathcal{S}}{\partial t_{u,k+1}} = 2 \cdot \sum_{i=1}^{n} (t_{u,k+1} - p_{iu} + \sum_{l=1}^{k} t_{ul} \cdot q_{il}) = 0$$
$$(u = 1, 2, \ldots, k).$$

We get the following system of linear equations:

$$\begin{pmatrix} a_{11} & \ldots & a_{1k} & b_1 & & & & & & & & \\ \vdots & \ddots & \vdots & \vdots & & & & & & & & \\ a_{k1} & \ldots & a_{kk} & b_k & & & & & & & & \\ b_1 & \ldots & b_k & n & & & & & & & & \\ & & & & a_{11} & \ldots & a_{1k} & b_1 & & & & \\ & & & & \vdots & \ddots & \vdots & \vdots & & & & \\ & & & & a_{k1} & \ldots & a_{kk} & b_k & & & & \\ & & & & b_1 & \ldots & b_k & n & & & & \\ & & & & & & & & \ddots & & & \\ & & & & & & & & & a_{11} & \ldots & a_{1k} & b_1 \\ & & & & & & & & & \vdots & \ddots & \vdots & \vdots \\ & & & & & & & & & a_{k1} & \ldots & a_{kk} & b_k \\ & & & & & & & & & b_1 & \ldots & b_k & n \end{pmatrix} \cdot \begin{pmatrix} t_{11} \\ \vdots \\ t_{1k} \\ t_{1,k+1} \\ t_{21} \\ \vdots \\ t_{2k} \\ t_{2,k+1} \\ \vdots \\ t_{k1} \\ \vdots \\ t_{kk} \\ t_{k,k+1} \end{pmatrix} = \begin{pmatrix} c_{11} \\ \vdots \\ c_{1k} \\ d_1 \\ c_{21} \\ \vdots \\ c_{2k} \\ d_2 \\ \vdots \\ c_{k1} \\ \vdots \\ c_{kk} \\ d_k \end{pmatrix},$$

where

$$a_{uv} = a_{vu} = \sum_{i=1}^{n} q_{iu} \cdot q_{iv}\,,$$

$$b_u = \sum_{i=1}^{n} q_{iu}\,,$$

$$c_{uv} = \sum_{i=1}^{n} p_{iu} \cdot q_{iv}\,,$$

$$d_u = \sum_{i=1}^{n} p_{iu}$$
$$(u = 1, 2, \ldots, k,\ v = 1, 2, \ldots, k).$$

The above system of linear equations can be solved by using an appropriate numerical method [9]. There exists a unique solution if and only if $det(M) \neq 0$, where

$$M = \begin{pmatrix} a_{11} & \ldots & a_{1k} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{k1} & \ldots & a_{kk} & b_k \\ b_1 & \ldots & b_k & n \end{pmatrix}.$$

Note that if a problem is close to singular (i.e., $det(M)$ is close to 0), the method can become unstable.

3 Discussion

In this section we state and prove a sufficient existence condition for a unique solution for the system of linear equations.

By a hyperplane of the Euclidean space $\mathbb{R}^k$ we mean a subset of the form $\{a + x : x \in S\}$ where S is a $(k-1)$-dimensional linear subspace. Given some points $q_1, \ldots, q_n$ in $\mathbb{R}^k$, we say that these points *span* $\mathbb{R}^k$ if no hyperplane of $\mathbb{R}^k$ contains them. If any $k+1$ points from $q_1, \ldots, q_n$ span $\mathbb{R}^k$ then we say that $q_1, \ldots, q_n$ are in *general position*.

Theorem 1. If $q_1, \ldots, q_n$ span $\mathbb{R}^k$ then $\det(M) \neq 0$.

Proof. Suppose $\det(M) = 0$. Consider the vectors $v_j = (q_{1j}, q_{2j}, \ldots, q_{nj})$ $(1 \leq j \leq k)$ in $\mathbb{R}^n$, and let $v_{k+1} = (1, 1, \ldots, 1) \in \mathbb{R}^n$. With the notation $m = k + 1$ observe that $M = \Big(\langle v_i, v_j \rangle\Big)_{m \times m}$ where $\langle\ ,\ \rangle$ stands for the scalar multiplication. Since the columns of M are linearly dependent, we can fix a $(\beta_1, \ldots, \beta_m) \in \mathbb{R}^m \setminus \{(0, \ldots, 0)\}$ such that $\sum_{j=1}^{m} \beta_j \langle v_i, v_j \rangle = 0$ holds for $i = 1, \ldots, m$. Then

$$0 = \sum_{i=1}^{m} \beta_i \cdot 0 = \sum_{i=1}^{m} \beta_i \sum_{j=1}^{m} \beta_j \langle v_i, v_j \rangle = \sum_{i=1}^{m} \beta_i \Big\langle v_i, \sum_{j=1}^{m} \beta_j v_j \Big\rangle =$$
$$\Big\langle \sum_{i=1}^{m} \beta_i v_i, \sum_{j=1}^{m} \beta_j v_j \Big\rangle = \Big\langle \sum_{i=1}^{m} \beta_i v_i, \sum_{i=1}^{m} \beta_i v_i \Big\rangle,$$

whence $\sum_{i=1}^{m} \beta_i v_i = 0$. Therefore all the q_j, $1 \leq j \leq n$, are solutions of the following (one element) system of linear equations:

$$\beta_1 x_1 + \cdots + \beta_k x_k = -\beta_m. \tag{1}$$

Since the system has solutions and $(\beta_1, \ldots, \beta_m) \neq (0, \ldots, 0)$, there is an $i \in \{1, \ldots, k\}$ with $\beta_i \neq 0$. Hence the solutions of (1) form a hyperplane of $\mathbb{R}^k$. This hyperplane contains $q_1, \ldots, q_n$. Now it follows that if $q_1, \ldots, q_n$ span $\mathbb{R}^k$ then $\det(M) \neq 0$. Q.e.d.

4 Conclusions

In real applications, it is assumed that both $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ span $\mathbb{R}^k$. Then, if the matching error is zero (i.e., $p'_i = T(q_i) = p_i$ for $i = 1, 2, \ldots n$), the transformation is necessarily non-degenerate, i.e., $\det(\mathcal{T}) \neq 0$. Moreover, in this case the following property is fulfilled:

Observation 2. For all $I \subseteq \{1, \ldots, n\}$ with $k+1$ elements, the p_i, $i \in I$, span $\mathbb{R}^k$ if and only if the q_i, $i \in I$, span $\mathbb{R}^k$.

This raises the question whether the transformation is necessarily non–degenarete in general or when Observation 2 holds or at least when Observation 2 "strongly" holds in the following computational sense: each simplex with vertices in $\{p_1, \ldots, p_n\}$ or with vertices in $\{q_1, \ldots, q_n\}$ has a large volume (k–dimensional measure) compared with its edges.

Surprisingly, all these questions have a negative answer, for we have the following three dimensional example.

Example 3. With $n = 5$ and $k = 3$ let $q_1 = (0, 0, 24)$, $q_2 = (24, 0, 0)$, $q_3 = (0, 24, 0)$, $q_4 = (0, 0, 0)$, and $q_5 = (-24, -48, 16)$. These five points determine five tetrahedra with reasonably large volumes, the smallest of them being 1536, the volume of the tetrahedron (q_2, q_3, q_4, q_5). Let $p_1 = (0, 0, 0)$, $p_2 = (3, 0, 0)$, $p_3 = (0, 3, 0)$, $p_4 = (0, 0, 3)$, $p_5 = (3, 3, 3)$, these are some vertices of a cube, so the tetrahedra they determine are at least of volume $9/2$. Yet,

$$\mathcal{T} = \begin{pmatrix} 2 & -6 & -6 & 12 \\ -9 & -1 & -9 & 18 \\ 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

which is degenerate.

Experience shows that in real applications the choice of points always guarantees that the transformation is non–degenerate [11, 12]. However, from theoretical point of view the following open problem is worth raising: Find a meaningful sufficient condition to ensure non–degeneracy.

Acknowledgement

This work was supported by OTKA T023804, OTKA T026243, OTKA T023186, and FKFP 0908/1997 grants.

References

[1] Arun, K.S., T.S. Huang, S.D. Blostein, Least squares fitting of two 3-D point sets, *IEEE Trans. Pattern Analysis and Machine Intelligence* **9** (1987), 698–703.

[2] Bookstein, F.L., Principal warps: thin-plate splines and the decomposition of deformations, *IEEE Trans. Pattern Analysis and Machine Intelligence* **11** (1989), 567–585.

[3] Faugeras, O.D., M. Hebert, A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces, Proc. Int. Joint Conf. Artificial Intelligence, Karlsruhe, 1983, 996–1002.

[4] Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes, *Computer Graphics — Principles and practice*, Addison–Wesley Publishing Company, Reading, Massachusetts 1991.

[5] Fornefett, M., K. Rohr, H.S. Stiehl, Radial basis functions with compact support for elastic registration of medical images, Proc. Int. Workshop on Biomedical Image Registration, Bled, 1999, 173–185.

[6] Horn, B.K.P., Closed-form solution of absolute orientation using unit quaternions, *J. Opt. Soc. Amer. A* **4** (1987), 629–642.

[7] Horn, B.K.P., Closed-form solution of absolute orientation using orthonormal matrices, *J. Opt. Soc. Amer. A* **5** (1987), 1127–1135.

[8] Maguire, D., M.F. Goodchild, D.W. Rhind (eds.), *Geographical information systems — Principles and Applications*, Longman Scientific and Technical, 1991.

[9] Press, W.H., B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1992.

[10] Sprinzak, J., M. Werman, Affine Point Matching, *Pattern Recognition Letters* **4** (1994), 337–339.

[11] Tanács, A., K. Palágyi, A. Kuba, Medical image registration based on interactively defined anatomical landmark points, *Int. J. Machine Graphics & Vision* **7** (1998), 151–158.

[12] Tanács, A., K. Palágyi, A. Kuba, Target Registration Error of Point–Based Methods Assuming Rigid–Body and Linear Motions, Proc. Int. Workshop on Biomedical Image Registration, Bled, 1999, 223–233.

[13] Umeyama, S., Least-squares estimation of transformation parameters between two point patterns, *IEEE Trans. Pattern Analysis and Machine Intelligence* **13** (1991), 376–380.

Received April, 2000

Acta Cybernetica **15** (2001) 107-117.

The complexity of coloring graphs without long induced paths

Gerhard J. Woeginger * Jiří Sgall †

Abstract

We discuss the computational complexity of determining the chromatic number of graphs without long induced paths. We prove NP-completeness of deciding whether a P_8-free graph is 5-colorable and of deciding whether a P_{12}-free graph is 4-colorable. Moreover, we give a polynomial time algorithm for deciding whether a P_5-free graph is 3-colorable.

Keywords: graph coloring – chromatic number – computational complexity – induced path.

1 Introduction

A graph $G = (V, E)$ is *k-colorable* if there exists a coloring $f : V \to \{1, \ldots, k\}$ such that $f(u) \neq f(v)$ for every edge $[u, v] \in E$. The *chromatic number* $\chi(G)$ of graph G is the smallest k for which G is k-colorable. A graph $G = (V, E)$ is *P_m-free* if it does not contain the path P_m on m vertices as an induced subgraph. For $v \in V$, we denote by $\Gamma(v) = \{w \in V : [v, w] \in E\}$ the neighborhood of v. For $W \subseteq V$, denote $\Gamma(W) = \bigcup_{w \in W} \Gamma(w)$.

In this note we discuss the computational complexity of deciding whether a given P_m-free graph G is k-colorable. For all $m \geq 2$ and $k \geq 2$, we call the corresponding coloring problem $P(m, k)$.

The problems $P(m, 2)$ are polynomially solvable, since 2-coloring is polynomially solvable even for arbitrary graphs; see e.g. Garey & Johnson [1]. Similarly, the problems of type $P(4, k)$ are polynomially solvable: A graph is P_4-free if and only if it is a cograph, and the chromatic number of a cograph can be determined in polynomial time; see Golumbic [2]. (The special cases $P(2, k)$ and $P(3, k)$ are trivial since P_3-free graphs are disjoint unions of cliques.)

In this note we will prove the following results.

*Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria, email:gwoegi@opt.math.tu-graz.ac.at. Supported by the START program Y43-MAT of the Austrian Ministry of Science.

†Mathematical Inst., AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, email: sgall@math.cas.cz. Partially supported by grant A1019901 of GA AV ČR, grant 201/01/1195 of GA ČR, and project LN00A056 of MŠMT CR.

Theorem 1.1 *It can be decided in polynomial time whether a P_5-free graph is 3-colorable.*

Theorem 1.2 *It is NP-complete to decide (a) whether a P_8-free graph is 5-colorable, and (b) whether a P_{12}-free graph is 4-colorable.*

These results and some of their implications are summarized in the table in Figure 1.

The proof of Theorem 1.1 is in Section 2, and the proofs of statements (a) and (b) in Theorem 1.2 are given in Sections 3 and 4, respectively.

m	4	5	6	7	8	9	10	11	12	13	14	15	...
$k=3$	P	P	??	??	??	??	??	??	??	??	??	??	...
$k=4$	P	??	??	??	??	??	??	??	NP	NP	NP	NP	...
$k=5$	P	??	??	??	NP	NP	NP	NP	NP	NP	NP	NP	...
$k=6$	P	??	??	??	NP	NP	NP	NP	NP	NP	NP	NP	...
$k=7$	P	??	??	??	NP	NP	NP	NP	NP	NP	NP	NP	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 1: A summary of complexity results for the coloring problems of type (m, k). An entry 'P' means that the problem is polynomially solvable, an entry 'NP' means that the problem is NP-hard, and an entry '??' means that the complexity of the problem is currently unknown.

2 The polynomial time result

In this section we prove Theorem 1.1. Consider a P_5-free graph $G = (V, E)$; without loss of generality we assume that G is connected. Our polynomial time algorithm distinguishes two cases for G; note that it is easy to distinguish the cases in time $O(n^3)$.

- Case 1: G is triangle-free. In this case we prove that the graph is 3-colorable and, in addition, we show how to construct a 3-coloring.
- Case 2: G contains a triangle. In this case we give an algorithm which reduces the problem to 2-satisfiability of propositional formulas.

Case 1. If G is bipartite, a 2-coloring is constructed easily. Otherwise G must contain an induced cycle of odd lengh. It cannot contain an induced cycle of length seven or more, since such an induced cycle would also yield an induced P_5. The case of C_3, i.e., a triangle is excluded in Case 1. Thus the induced cycle is C_5; denote its vertices v_0, v_1, v_2, v_3, v_4 in this ordering along the cycle. Denote $V_0 = \{v_0, v_1, v_2, v_3, v_4\}$.

Consider any vertex x adjacent to a vertex in V_0. If x is adjacent to only a single vertex from V_0, say to v_0, then x, v_0, v_1, v_2, v_3 would form an induced P_5 in G. If x is adjacent to two adjacent vertices from V_0, then these three vertices would form a triangle in G. As a consequence, x must have exactly two neighbors in V_0, and these two neighbors are not adjacent to each other. Denote by W_i, $0 \leq i \leq 4$, the set of all vertices in $V - V_0$ that are adjacent to v_{i-1} and v_{i+1} (all indices are taken modulo 5).

Now we state two simple observations. Our first observation is that $V = V_0 \cup W_0 \cup W_1 \cup W_2 \cup W_3 \cup W_4$. If not, since G is connected, there exists a vertex x not adjacent to V_0 but adjacent to some $y \in W_i$. But then $x, y, v_{i+1}, v_{i+2}, v_{i+3}$ is an induced P_4. Our second observation is that two distinct vertices x and y in some set $W_i \cup W_{i+2}$ cannot be adjacent to each other, since this would yield a triangle x, y, v_{i+1}.

Consequently G can be 3-colored by coloring all vertices in W_0 and W_2 by 1, all vertices in W_1 and W_3 by 2, and all vertices in W_3 by 3. Moreover, the partition W_0, W_1, W_2, W_3, W_4 can be computed in polynomial time.

Case 2. Whereas all graphs in the first case were 3-colorable, the second case covers several graphs that are not 3-colorable, for example K_4 (the complete graph on 4 vertices) or $C_5 + K_1$ (the graph that results from connecting a new vertex to all vertices of a cycle on five vertices).

Consider an arbitrary triangle in G and color its vertices by 1, 2, 3. As long as there is an uncolored vertex v that has neighbors of two different colors, color v with the remaining third color. Note that all these moves are forced. If we find an uncolored vertex that has neighbors of three different colors, then we conclude that G is not 3-colorable. When this process terminates, we denote by V_0 the set of all colored vertices, we denote by V_1 (respectively, V_2 and V_3) the set of all uncolored vertices that are adjacent to some colored vertex of color 1 (respectively, color 2 and 3). Furthermore, we denote by V_4 the set of all vertices in $V - V_0 \cup V_1 \cup V_2 \cup V_3$ that are adjacent to some vertex in $V_1 \cup V_2 \cup V_3$.

We claim that $V = V_0 \cup V_1 \cup V_2 \cup V_3 \cup V_4$. Suppose otherwise. Then since G is connected, there must exist some vertex x outside of $V_0 \cup V_1 \cup V_2 \cup V_3 \cup V_4$ that is adjacent to some vertex y in V_4. Vertex y is adjacent to some vertex in $V_1 \cup V_2 \cup V_3$, say to vertex $z \in V_1$. Vertex z is adjacent to a 1-colored vertex v_1 in V_0, and v_1 is adjacent to a 2-colored vertex v_2 in V_0. But then x, y, z, v_1, v_2 form an induced P_5 in G. This contradiction shows that indeed $V = V_0 \cup V_1 \cup V_2 \cup V_3 \cup V_4$.

Lemma 2.1 *Let $C = \{x\}$ be a connected component in V_4 that consists of a single vertex. If the graph G is 3-colorable, then one of the following two situations holds.*

(i) *$\Gamma(x) \subseteq V_i$ for some i with $1 \leq i \leq 3$. Any 3-coloring of $V_0 \cup V_1 \cup V_2 \cup V_3$ uses at most two colors on $\Gamma(x) \subseteq V_i$. Any such 3-coloring can be extended to vertex x.*

(ii) *$\Gamma(x) \cap V_i \neq \emptyset$ and $\Gamma(x) \cap V_j \neq \emptyset$ for some i, j with $1 \leq i < j \leq 3$. In this case $\Gamma(x)$ forms a complete bipartite subgraph of G with bipartition $\Gamma(x) \cap V_i$*

and $\Gamma(x) \cap V_j$. In any 3-coloring of G, all vertices in $\Gamma(x) \cap V_i$ must receive the same color and all vertices in $\Gamma(x) \cap V_j$ must receive the same color. Any such 3-coloring can be extended to vertex x.

Proof. First suppose that there are two vertices $w_1 \in V_1 \cap \Gamma(x)$ and $w_2 \in V_2 \cap \Gamma(x)$ such that $[w_1, w_2] \notin E$. Then w_1 is adjacent to a 1-colored vertex v_1 in V_0, and this vertex v_1 is adjacent to a 3-colored vertex v_3 in V_0. But then w_2, x, w_1, v_1, v_3 would form an induced P_5 in G. As a consequence, any two vertices in $\Gamma(x)$ that do not form an edge in G must both belong to the same set V_1, V_2, V_3.

We now consider several cases. First assume that $\Gamma(x)$ intersects all three sets V_1, V_2, and V_3. Then three vertices in $\Gamma(x) \cap V_1$, $\Gamma(x) \cap V_2$ and $\Gamma(x) \cap V_3$ together with x would form a K_4, and G would not be 3-colorable in this case. Next assume that $\Gamma(x) \cap V_i \neq \emptyset$ and $\Gamma(x) \cap V_j \neq \emptyset$ for some i, j with $1 \leq i < j \leq 3$. By the observation in the preceding paragraph, G must contain all edges between $\Gamma(x) \cap V_i$ and $\Gamma(x) \cap V_j$. Then $\Gamma(x) \cap V_i$ and $\Gamma(x) \cap V_j$ must both be independent sets, since otherwise G would contain a K_4. Hence, in this case we are in situation (ii). Finally, $\Gamma(x) \subseteq V_i$ might hold for some i with $1 \leq i \leq 3$ as in situation (i). □

Lemma 2.2 *Let C be a connected component in V_4 that contains at least two vertices. Denote the set $\Gamma(C) \cap (V_1 \cup V_2 \cup V_3)$ by D.*

(i) *All vertices in C are adjacent to all vertices in D.*

(ii) *If the graph G is 3-colorable, then the component C is bipartite and D forms an independent set.*

(iii) *In any 3-coloring of G, all vertices in D receive the same color.*

(iv) *Assume that C is bipartite. Then any 3-coloring of $V_0 \cup V_1 \cup V_2 \cup V_3$ in which all vertices in D have the same color can be extended to a 3-coloring of $V_0 \cup V_1 \cup V_2 \cup V_3$ and C.*

Proof. (i) Let $x, y \in C$ with $[x, y] \in E$. Suppose that there exists some $z \in D$ that is adjacent to x but not to y; without loss of generality $z \in V_1$. Then z is adjacent to some 1-colored v_1 in V_0, and v_1 is adjacent to a 2-colored vertex v_2 in V_0. Then x, y, z, v_1, v_2 form an induced P_5 in G. This contradiction shows that x and y have exactly the same neighbors in D. This yields statement (i).

Statement (ii) is an immediate consequence of (i): Any 3-coloring of G must color the component C with two colors, and the neighborhood D of C with the third color. This also yields statement (iii). Statement (iv) is straightforward. □

With the help of Lemmas 2.1 and 2.2, we will now translate the 3-coloring problem into a TWO-SATISFIABILITY problem. Since the colors of vertices in V_0 have already been fixed, we will concentrate on the vertices in V_1, V_2, and V_3.

- For every $v \in V_i$ $(1 \leq i \leq 3)$ we introduce two variables $x(v, j)$ and $x(v, k)$ such that $\{i, j, k\} = \{1, 2, 3\}$. A TRUE variable $x(v, j)$ will mean that vertex

v is colored by color j. By introducing two clauses with two literals, we can enforce that exactly one of $x(v,j)$ and $x(v,k)$ must be TRUE and the other one must be false.

- Consider a connected component $C = \{x\}$ in V_4. If neither situation (i) nor (ii) from Lemma 2.1 hold, then we stop the construction since G is not 3-colorable. If situation (i) holds, we do not need to do anything. And if situation (ii) holds, then we introduce several clauses (each with two literals) that enforce that all vertices in $\Gamma(x) \cap V_i$ get the same color and that all vertices in $\Gamma(x) \cap V_j$ get the same color.

- Consider a connected component C in V_4 that contains at least two vertices. If the neighbors of C in $V_1 \cup V_2 \cup V_3$ do not form an independent set, then we stop the construction since G is not 3-colorable by Lemma 2.2. And if they do form an independent set, then we introduce several clauses (each with two literals) that enforce that all these vertices get the same color.

The resulting instance of TWO-SATISFIABILITY can be solved in polynomial time; see e.g. Garey & Johnson [1]. If this TWO-SATISFIABILITY does not have a satisfying truth assignment, then by Lemmas 2.1 and 2.2 the graph G cannot be 3-colorable. On the other hand, if this TWO-SATISFIABILITY does have a satisfying truth assignment, then we can translate it into a 3-coloring for $V_0 \cup V_1 \cup V_2 \cup V_3$ and we can use Lemmas 2.1 and 2.2 to extend this coloring to a 3-coloring for V_4. Since all this can clearly be done in polynomial time, the proof of Theorem 1.1 is complete.

3 The first NP-hardness proof

In this section we prove Theorem 1.2(a). The reduction is from the NP-hard THREE-SATISFIABILITY problem (Garey & Johnson [1]): Given a set $X = \{x_1, \ldots, x_n\}$ of logical variables, and a set $C = \{c_1, \ldots, c_m\}$ of three-literal clauses over X, does there exist a truth assignment for X that simultaneously satisfies all clauses in C?

Now consider an arbitrary instance I of THREE-SATISFIABILITY. We define a P_8-free graph $G_1 = (V_1, E_1)$ that is 5-colorable if and only if this instance I has answer YES:

- For every variable $x \in X$, there is a vertex $a(x)$ that corresponds to the unnegated literal x, and a vertex $a(\overline{x})$ that corresponds to the negated literal $\overline{x}$. These vertices are connected to each other by an edge.

- For every clause $c \in C$ that consists of the literals u_1, u_2, u_3, there are seven corresponding vertices $b_1(c)$, $b_2(c)$, $b_3(c)$, $b_4(c)$ and $b(c,u_1)$, $b(c,u_2)$, $b(c,u_3)$. The three vertices $b_1(c)$, $b_2(c)$, $b_3(c)$ form a triangle. Moreover, $b_1(c)$ is connected to $b(c,u_2)$ and $b(c,u_3)$, $b_2(c)$ is connected to $b(c,u_1)$ and $b(c,u_3)$, and $b_3(c)$ is connected to $b(c,u_1)$ and $b(c,u_2)$. Vertex $b_4(c)$ is connected to

$b(c,u_1)$, $b(c,u_2)$, $b(c,u_3)$. For $i = 1,2,3$ vertex $b(c,u_i)$ is connected to the vertex $a(u_i)$. See Figure 2 for an illustration.

- All vertices $a(x)$ and $a(\overline{x})$ with $x \in X$ are connected to all vertices $b_1(c)$, $b_2(c)$, $b_3(c)$, $b_4(c)$ with $c \in C$.
- Finally, there is a single dummy vertex d that is connected to all clause vertices $b_1(c)$, $b_2(c)$, $b_3(c)$, $b_4(c)$ and $b(c,u_1)$, $b(c,u_2)$, $b(c,u_3)$ with $c = u_1 \vee u_2 \vee u_3$ in C.

This completes the description of the graph G_1. Note that G_1 may contain an induced P_7 that runs through $a(\overline{x_1})$, $a(x_1)$, $b(c_2,x_1)$, d, $b(c_3,x_4)$, $a(x_4)$, $a(\overline{x_4})$.

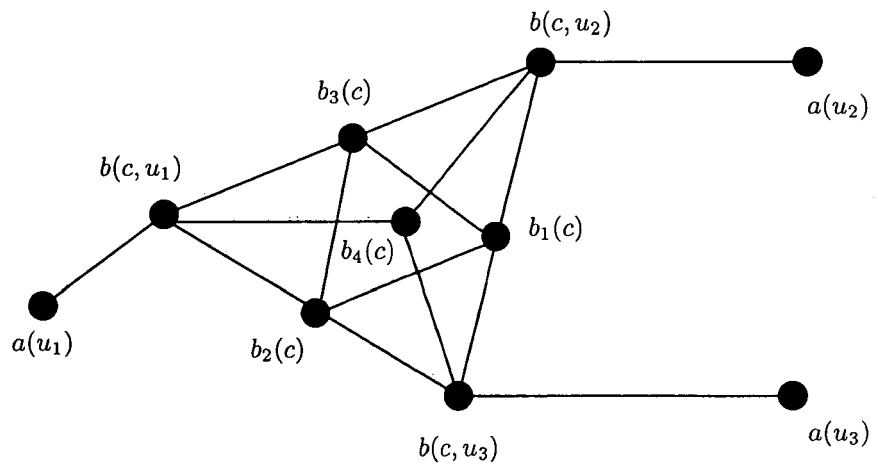


Figure 2: The seven vertex gadget for a clause in graph G_1.

Lemma 3.1 *The graph G_1 is P_8-free.*

Proof. Suppose that to the contrary G_1 would contain an induced path P on eight vertices. Denote by A the set of all vertices $a(u_i)$ on P, denote by B_1 the set of vertices $b_h(c)$ on P, and denote by B_2 the set of vertices $b(c,u_j)$ on P. We start with three observations.

(1) First, suppose that P contains the dummy vertex d. Since d is adjacent to all vertices in B_1 and in B_2, this yields $|B_1 \cup B_2| \leq 2$ and hence $|A| \geq 5$. Since $\{d\} \cup B_1 \cup B_2$ is a connected set, removing it from P decomposes this (induced) path into at most two (induced) subpaths that both are spanned by A. Since $|A| \geq 5$, one of these two subpaths must contain at least three vertices. But the longest induced paths in A have only two vertices. This contradiction yields $d \notin P$.

(2) Next suppose that $A = \emptyset$. Then P only contains vertices from B_1 and B_2. But the longest induced path in $B_1 \cup B_2$ has only four vertices; cf. Figure 2. This contradiction shows $|A| \geq 1$.

(3) Next suppose that $|A| \geq 3$. Since all vertices in B_1 are adjacent to all vertices in A, this would imply $B_1 = \emptyset$. Then P only contains vertices from A and B_2. But the longest induced path in $A \cup B_2$ has only four vertices, and this shows $|A| \leq 2$.

The observations in (2) and (3) yield $1 \leq |A| \leq 2$. Since all vertices in B_1 are adjacent to all vertices in A, $|A| = 1$ implies $|B_1| \leq 2$, and $|A| = 2$ implies $|B_1| \leq 1$. Therefore $|A \cup B_1| \leq 3$ and $|B_2| \geq 5$. But B_2 is an independent set of at least five vertices and cannot be connected to an induced path by adding the at most three vertices from $A \cup B_1$. □

Lemma 3.2 *If the THREE-SATISFIABILITY instance I has a satisfying truth assignment, then the graph G_1 is 5-colorable.*

Proof. We define a coloring from the truth assignment. If $x \in X$ is TRUE then color $a(x)$ by 4 and $a(\overline{x})$ by 5, and if x is FALSE then color $a(x)$ by 5, and $a(\overline{x})$ by 4. The dummy vertex d receives color 4. Now consider the seven vertices $b_1(c)$, $b_2(c)$, $b_3(c)$, $b_4(c)$ and $b(c,u_1)$, $b(c,u_2)$, $b(c,u_3)$, that correspond to a clause $c = u_1 \vee u_2 \vee u_3$ in C. One of the three literals u_1, u_2, u_3 must be true, and so we may assume without loss of generality that u_1 is a true literal, and that hence $a(u_1)$ is colored 4. In this case we color $b(c,u_1)$ by 5, $b(c,u_2)$ by 2, $b(c,u_3)$ by 3, and $b_1(c)$ by 1, $b_2(c)$ by 2, $b_3(c)$ by 3, $b_4(c)$ by 1. This coloring is legal: the edges among the seven clause vertices are verified easily, and besides them $b(c,u_1)$ is adjacent only to vertices already colored by 4 and the other six vertices are adjacent only to vertices already colored by 4 and 5. The cases where u_2 or u_3 are true literals are handled analogously. □

Lemma 3.3 *If the graph G_1 is 5-colorable, then the THREE-SATISFIABILITY instance I has a satisfying truth assignment.*

Proof. Consider an arbitrary triangle $b_1(c)$, $b_2(c)$, $b_3(c)$ in a clause gadget. Without loss of generality, these three vertices are colored by colors 1, 2, and 3. Since the triangle vertices are adjacent to the dummy vertex d and to all literal vertices $a(x)$ and $a(\overline{x})$, all these adjacent vertices must be colored by 4 or by 5. Without loss of generality assume that the dummy vertex has color 4. Furthermore, if $a(x)$ has color 4, then $a(\overline{x})$ has color 5, and if $a(x)$ has color 5, then $a(\overline{x})$ has color 4. Define a truth assignment for X that sets variable x to TRUE if and only if $a(x)$ has color 4.

Suppose that some clause $c = u_1 \vee u_2 \vee u_3$ in C is not satisfied under this truth assignment. Then the three vertices $a(u_1)$, $a(u_2)$, $a(u_3)$ all are colored 5. Then the three clause vertices $b(c,u_1)$, $b(c,u_2)$, $b(c,u_3)$ have a neighbor of color 5, and the dummy vertex as neighbor of color 4, and hence they must be colored by colors 1,

2, 3. The four clause vertices $b_1(c)$, $b_2(c)$, $b_3(c)$, $b_4(c)$ are adjacent to $a(u_1)$ and $a(\overline{u_1})$ of colors 4 and 5, and hence they must be colored by colors 1, 2, 3. This implies that the seven vertices of the clause gadget are legally colored by the three colors 1, 2, 3, which clearly is impossible. This contradicts our assumption that the coloring is legal, and therefore the constructed truth assignment indeed satisfies all clauses. □

The three Lemmas 3.1, 3.2, and 3.3 together prove Theorem 1.2(a).

4 The second NP-hardness proof

In this section we prove Theorem 1.2(b). Again, the reduction is from the NP-hard THREE-SATISFIABILITY problem; cf. the first paragraph of the preceding section. Consider an arbitrary instance I of THREE-SATISFIABILITY that consists of a set $X = \{x_1, \ldots, x_n\}$ of logical variables, and a set $C = \{c_1, \ldots, c_m\}$ of three-literal clauses over X. We will define a P_{12}-free graph $G_2 = (V_2, E_2)$ that is 4-colorable if and only if this instance I of THREE-SATISFIABILITY has answer YES.

- For every variable $x \in X$, there is a vertex $a(x)$ that corresponds to the unnegated literal x, and a vertex $a(\overline{x})$ that corresponds to the negated literal $\overline{x}$. These vertices are connected to each other by an edge.
- For every clause $c \in C$ that consists of the literals u_1, u_2, u_3, there are nine corresponding vertices $b_1(c,u_1)$, $b_1(c,u_2)$, $b_1(c,u_3)$, and $b_2(c,u_1)$, $b_2(c,u_2)$, $b_2(c,u_3)$, and $b_3(c,u_1)$, $b_3(c,u_2)$, $b_3(c,u_3)$. The three vertices $b_1(c,u_1)$, $b_1(c,u_2)$, $b_1(c,u_3)$ form a triangle. Moreover, for $i = 1,2,3$ the vertex $b_1(c,u_i)$ is connected to $b_2(c,u_i)$, the vertex $b_2(c,u_i)$ is connected to $b_3(c,u_i)$, and the vertex $b_3(c,u_i)$ is connected to $a(u_i)$. See Figure 3 for an illustration.
- All vertices $b_2(c,u_i)$ are connected to all vertices $a(x)$ and $a(\overline{x})$.
- Finally, there are two dummy vertices d_1 and d_2 that are connected to each other by an edge. The dummy vertex d_1 is connected to all vertices $b_j(c,u_i)$ that belong to the clause gadgets. The dummy vertex d_2 is connected to all vertices $a(x)$ and $a(\overline{x})$ with $x \in X$, and to all vertices $b_3(c,u_i)$.

This completes the description of the graph G_2. Note that G_2 may contain an induced P_{11} that runs through $b_2(c,u_4)$, $b_1(c,u_4)$, $b_1(c,u_5)$, $b_2(c,u_5)$, $b_3(c,u_5)$, d_2, $b_3(c',u_6)$, $b_2(c',u_6)$, $b_1(c',u_6)$, $b_1(c',u_7)$, $b_2(c',u_7)$.

Lemma 4.1 *The graph G_2 is P_{12}-free.*

Proof. Suppose that to the contrary G_2 would contain an induced path P on twelve vertices. Denote by A the set of all vertices $a(u_i)$ on P, denote by D the set of dummy vertices on P, and for $h = 1,2,3$ denote by B_h the set of all vertices $b_h(c,u_i)$ on P. We start with four simple observations.

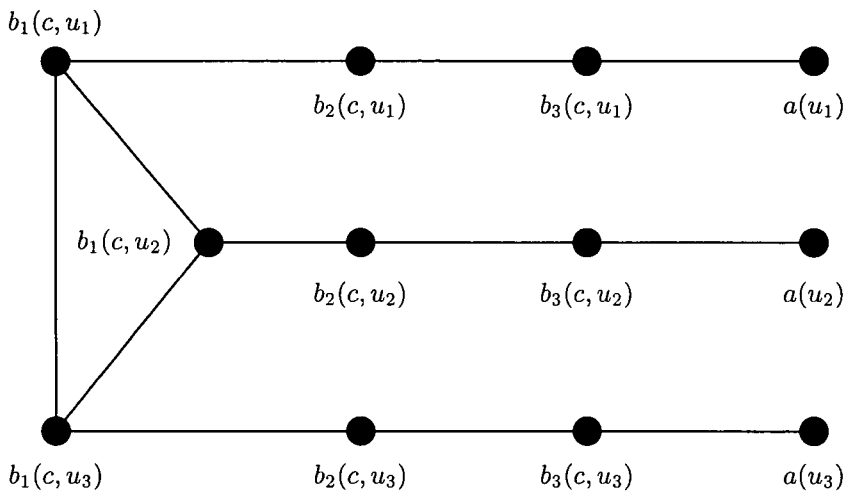


Figure 3: The nine vertex gadget for a clause in graph G_2.

(1) First, suppose that $|A| \geq 3$. Since all vertices in B_2 are adjacent to all vertices in A, this would imply $B_2 = \emptyset$. Moreover, $d_2 \notin D$. Then the only connection from B_1 to the rest of P is via the dummy vertex d_1, and therefore the vertices in $B_3 \cup A$ must induce a path. But there clearly cannot be an induced path in $B_3 \cup A$ that contains three or more vertices from A. This contradiction shows $|A| \leq 2$.

(2) Next suppose that $d_1 \in D$. Since d_1 is adjacent to all vertices in $B_1 \cup B_2 \cup B_3$, this union $B_1 \cup B_2 \cup B_3$ contains at most two elements. But then $|D \cup A| \geq 10$ which contradicts (1). Hence, $d_1 \notin D$.

(3) Suppose that $d_2 \in D$. Since d_2 is adjacent to all vertices in $B_3 \cup A$, we have $|B_3 \cup A| \leq 2$ and $|B_1 \cup B_2| \geq 9$. Every induced path in $B_1 \cup B_2$ has at most four vertices, and thus $B_1 \cup B_2$ must induce at least three subpaths of P. Each of these subpaths needs one adjacent vertex in B_3, which contradicts $|B_3| \leq |B_3 \cup A| \leq 2$. Hence, $d_2 \notin D$.

(4) Next suppose that $A = \emptyset$. Then P only contains vertices from B_1, B_2, and B_3. But the longest induced path in $B_1 \cup B_2 \cup B_3$ has only six vertices; cf. Figure 3. Therefore, $|A| \geq 1$.

The observations in (1) and (4) yield $1 \leq |A| \leq 2$. Since all vertices in B_2 are adjacent to all vertices in A, $|A| = 1$ implies $|B_2| \leq 2$, and $|A| = 2$ implies $|B_2| \leq 1$. Therefore $|A \cup B_2| \leq 3$, and $|B_1 \cup B_3| \geq 9$. But the longest induced paths in $B_1 \cup B_3$ have only two vertices, and thus $B_1 \cup B_3$ induce at least five connected components.

There is no way of glueing these at least five components together via the at most three vertices in $A \cup B_2$. □

Lemma 4.2 *If the THREE-SATISFIABILITY instance I has a satisfying truth assignment, then the graph G_2 is 4-colorable.*

Proof. We define a coloring from the truth assignment. Dummy vertex d_1 is colored 1, and dummy vertex d_2 is colored 2. All vertices $a(u_i)$ will be colored 1 or 4; all vertices $b_3(c, u_i)$ will be colored 3 or 4; all vertices $b_2(c, u_i)$ will be colored 2 or 3; all vertices $b_1(c, u_i)$ will be colored 2, 3, or 4. Clearly, by doing so we will avoid all color conflicts with the two dummy vertices. Moreover, vertices $b_2(c, u_i)$ will receive other colors than vertices $a(x)$ and $a(\overline{x})$.

If $x \in X$ is TRUE then color $a(x)$ by 1 and $a(\overline{x})$ by 4, and if x is FALSE then color $a(x)$ by 4, and $a(\overline{x})$ by 1. Now consider the vertices of some clause gadget for $c = u_1 \vee u_2 \vee u_3$ in C. Without loss of generality we assume that the literal u_1 is true, and that hence $a(u_1)$ is colored 1. Then we color

$b_1(c, u_1)$ by 2	$b_2(c, u_1)$ by 3	$b_3(c, u_1)$ by 4	$a(u_1)$ is 1
$b_1(c, u_2)$ by 3	$b_2(c, u_2)$ by 2	$b_3(c, u_2)$ by 3	$a(u_2)$ is 1 or 4
$b_1(c, u_3)$ by 4	$b_2(c, u_3)$ by 2	$b_3(c, u_3)$ by 3	$a(u_3)$ is 1 or 4

It is easy to verify that we indeed end up with a legal 4-coloring for the graph G_2. □

Lemma 4.3 *If the graph G_2 is 4-colorable, then the THREE-SATISFIABILITY instance I has a satisfying truth assignment.*

Proof. Without loss of generality we assume that in the 4-coloring dummy vertex d_1 is colored 1 and that dummy vertex d_2 is colored 2. The set of literal vertices $a(x)$ and $a(\overline{x})$ uses at least two different colors. We claim that also the set of clause vertices $b_2(c, u_i)$ uses at least two different colors: Suppose that to the contrary all vertices $b_2(c, u_i)$ are colored by a single color, say, by color 3. Then the triangles $b_1(c, u_1)$, $b_1(c, u_2)$, $b_1(c, u_3)$ cannot use this color 3, nor can they use the color 1 of dummy vertex d_1. But it is impossible to color the triangle legally by colors 2 and 4 only, which proves our claim. Now since the literal vertices use at least two colors, since the vertices $b_2(c, u_i)$ use at least two colors, and since these two vertex classes form a complete bipartite graph, one of these classes must use exactly two colors, and the other class must use the remaining two colors. Without loss of generality we assume that the literal vertices use the colors 1 and 4, and that the vertices $b_2(c, u_i)$ use the colors 2 and 3.

We define a truth assignment from the 4-coloring by setting variable x to TRUE if and only if vertex $a(x)$ has color 1. Suppose that some clause $c = u_1 \vee u_2 \vee u_3$ in C is not satisfied under this truth assignment. Then the three vertices $a(u_1)$, $a(u_2)$, $a(u_3)$ all are colored 4. The three adjacent vertices $b_3(c, u_1)$, $b_3(c, u_2)$, $b_3(c, u_3)$ cannot use color 4, and they also cannot use colors 1 or 2 since they are adjacent to both dummy vertices; therefore, $b_3(c, u_1)$, $b_3(c, u_2)$, $b_3(c, u_3)$ all are colored 3. Then the three adjacent vertices $b_2(c, u_1)$, $b_2(c, u_2)$, $b_2(c, u_3)$ cannot use this color 3; by

the above discussion all three vertices $b_2(c,u_1)$, $b_2(c,u_2)$, $b_2(c,u_3)$ must be colored 2. Then the three adjacent vertices $b_1(c,u_1)$, $b_1(c,u_2)$, $b_1(c,u_3)$ cannot use this color 2, and they also cannot use the color 1 of the adjacent dummy vertex d_1. This implies that the triangle $b_1(c,u_1)$, $b_1(c,u_2)$, $b_1(c,u_3)$ is legally colored by the two colors 3 and 4, which is impossible. Consequently, the constructed truth assignment satisfies all clauses in C. □

The three Lemmas 4.1, 4.2, and 4.3 together prove Theorem 1.2(b).

References

[1] M.R. Garey and D.S. Johnson [1979]. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

[2] M.C. Golumbic [1980]. *Algorithmic Graph Thery and Perfect Graphs.* Academic Press, New York.

[3] D.S. Johnson [1985]. The NP-completeness column: An ongoing guide. *Journal of Algorithms 6,* 434–451.

Received Rebruary, 2001

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 2001. június